April 1990 UILU-ENG-90-2210

2

COORDINATED SCIENCE LABORATORY
College of Engineering

THROUGHPUT ANALYSIS OF SCHEDULING ALGORITHMS FOR A TDMA SILENT RECEIVER SYSTEM

Dominic Michael Tolli

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

UNCLASSIFIED
~~SECURITY CLASSIFICATION OF THIS PAGE~~

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-90-2210	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois — 6b. OFFICE SYMBOL *(If applicable)* N/A	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research
6c. ADDRESS *(City, State, and ZIP Code)* 1101 W. Springfield Ave. Urbana, IL 61801	7b. ADDRESS *(City, State, and ZIP Code)* Arlington, VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Joint Services Electronics Program — 8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-90-J-1270
8c. ADDRESS *(City, State, and ZIP Code)* Arlington, VA 22217	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*
Throughput Analysis of Scheduling Algorithms for a TDMA Silent Receiver System

12. PERSONAL AUTHOR(S)
Tolli, Dominic Michael

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
Technical	FROM ______ TO ______	April, 1990	52

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
scheduling algorithms, spread spectrum, incremental redundancy; radio networks

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

Several methods of scheduling a time-division multiple access, or TDMA, receiver with no feedback capability are evaluated for their throughput performance. Some of the options considered include prior time as opposed to real time scheduling, optimal as opposed to simple or greedy scheduling, multiple listening capability, and the ability to utilize incremental redundancy techniques. All of these methods are compared by using various values for the rebroadcast redundancy. Keywords

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE *(Include Area Code)* — 22c. OFFICE SYMBOL

DD Form 1473, JUN 86 — *Previous editions are obsolete.* — SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

THROUGHPUT ANALYSIS OF SCHEDULING ALGORITHMS FOR A TDMA SILENT RECEIVER SYSTEM

BY

DOMINIC MICHAEL TOLLI

B.S., University of Illinois, 1988

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1990

Urbana, Illinois

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

THE GRADUATE COLLEGE

MAY 1990

WE HEREBY RECOMMEND THAT THE THESIS BY

DOMINIC MICHAEL TOLLI

ENTITLED THROUGHPUT ANALYSIS OF SCHEDULING ALGORITHMS FOR A TDMA SILENT RECEIVER SYSTEM

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF MASTER OF SCIENCE

[illegible]
Director of Thesis Research

N. Narayana Rao
Head of Department

Committee on Final Examination†

Chairperson

† Required for doctor's degree but not for master's.

O-517

ABSTRACT

Several methods of scheduling a time-division multiple access, or TDMA, receiver with no feedback capability are evaluated for their throughput performance. Some of the options considered include prior time as opposed to real time scheduling, optimal as opposed to simple or greedy scheduling, multiple listening capability, and the ability to utilize incremental redundancy techniques. All of these methods are compared by using various values for the rebroadcast redundancy.

ACKNOWLEDGEMENTS

I would like to thank the members of the University of Illinois Communication Group, whose assistance throughout the various stages of this research helped me to complete my work as efficiently as possible. I would especially like to thank my graduate advisor, Professor B. Hajek, whose patience and clear guidance made my research both enjoyable and rewarding.

Finally, I must express my gratitude to my family, especially my parents, whose care and endless accommodation allowed my work, both at home and at school, to be completed through many obstacles and pressures.

TABLE OF CONTENTS

I. INTRODUCTION

The basic problem considered in this thesis arises from the need to schedule the demand for a set of αM consumers to a set of M resource locations, as described in [1]. There are several different ways to define a desirable schedule, due to the many different strategies that may be employed to schedule consumers to resources, and also due to the different measures of performance that can be used. These strategies and measures need to be defined in a clear way to effectively compare scheduling algorithms.

This thesis will view this problem in the sense of a digital communication example, where channel access is desired by a set of consumers, known as the *users*, whose demand on the channel is bursty. The manner in which the channel is accessed by the users will be considered to be periodic in M [2], so that an assignment for access need only be constructed for M resource locations, which will be represented by *slots* of a channel time frame. For some integer Q, each user chooses Q random slots in the frame in which to broadcast its packet, forming what will be referred to as the *pre-assignment*. The number Q, called the "packet transmission diversity" in other papers, is referred to here as the *rebroadcast redundancy*. Each user then may only be scheduled to a subset of these pre-assigned slots. If the users employ the selective reception capability of spread spectrum signaling, this subset of users that may compete for and win access to the slots is seen to be a problem in efficiently scheduling a channel to users along a time frame [9]. Slow frequency hopping is the method of spread spectrum used in this thesis, and it is assumed that there are random, independent hopping patterns for the various codes used by each of the users. This frame-based contention for slots is then called "time-division multiple access," or TDMA.

No communication is needed from the channel to the users in order to schedule desirably, and, in fact, a *silent receiver* assumption may be viewed as a case where the users simply include their broadcast information in their packets so that the receiver can then schedule the users effectively. This assumption can occur in several situations of communication networks, as in [9]. For instance, the transmitters must be able to broadcast, but they may not have the ability to receive information due to space or cost limitations. Also, this can occur when the environment demands radio silence at the receiver. There may also be large levels of transmitter interference in the vicinity of some users, making available feedback unreliable for reception. As in [1], this scheduling method finds use in the problem of global load balancing of networks.

Primary and secondary conflicts of assignment for this problem may then arise [2]. A *primary conflict* occurs when more than one user is pre-assigned to a given slot. Problems with this sort of conflict can be avoided by scheduling only one user in any slot. These types of conflicts are the main types that are explored through the different methods of scheduling considered in this thesis, though secondary conflicts will also play a role in parts of this work. *Secondary conflicts* occur when more than one user broadcasts in a given slot, causing inter-user interference. These occur together with the primary conflicts due to the no-feedback assumption above which prevents the changing of a pre-assignment to eliminate conflicts. If a primary conflict does not occur originally, then neither can a secondary conflict. However, secondary conflicts are avoided by using the aforementioned spread spectrum signaling, which acts to distinguish a chosen user among several other users depending upon the total level of interference. Additionally, error control coding may be used to recover the errors caused by transmission of multiple users in the same spread spectrum frequency bin.

As a prelude to the modeling of these scheduling methods described in the next chapter, note how this basic problem may be seen as a case of bipartite graph matching. Figure 1 shows this description. On the top side, labelled $\underline{V}$, the nodes v_j represent the slots of the frame, and on the other side, labelled $\underline{U}$, the nodes u_i represent the various users wishing connection to the slots. Each graph edge represents a user's broadcast to a given slot, and the collection of these edges represents the pre-assignment. Each bold edge represents the scheduling of a given user to a given slot, so that the collection of these edges represents the chosen schedule. Note that each user has chosen Q=2 slots in which to broadcast, but not all the slots have the same number of pre-assigned users, indicating differing levels of other-user interference for each slot. Also indicated are edge weights, c_{ij}, representing a "cost" of scheduling user i to slot j. These weights are used in scheduling situations where it is desirable to consider overall costs of certain schedules as a sense of optimality.

The different scheduling strategies considered in this thesis are summarized in Table 1. The first distinction made is between "real time" and "prior time" scheduling. In *real time* scheduling, each slot is considered and scheduled according to information gathered about receptions in earlier slots. This information may include success or failure of transmission, and quality of transmission. Scheduling starts at the first frame slot and moves along the frame, so that slots later in the frame can be scheduled according to the performance of broadcasts earlier in the frame. In *prior time* scheduling, only the information about the network that is available prior to the beginning of the frame is used to form the schedule--it is not adapted to information gathered during the actual broadcasting. All of the systems analyzed in this thesis involve prior time scheduling, leaving the analysis of the real time algorithms to future study. Some discussion of the real time algorithms is included for reference.

Another distinction made is between "single" and "combining" reception. *Single reception* means that in a given frame, each scheduled user is scheduled in only one slot, stressing that as many users as possible be scheduled. The *combining reception* strategy allows a user to be scheduled in several slots, stressing that as many slots as possible be scheduled. A further distinction may be made in the latter case among methods of considering the several slots in which a user is heard. The first case is called *traditional retransmission*, and it involves sending the user's packet with the same error correction coding in each slot in which it is assigned. The decoder then receives the same redundant information each time the packet is received. The second case is known as *incremental redundancy*, a method in which all of the Q packets sent by a user in a given frame are coded as one "super-packet," and then decoded in each slot incrementally according to how many times the packet has already been received [3]. This latter case thus uses differing redundant information in each slot in which a user broadcasts, so that a throughput increase may be achieved.

Note that for the single reception case, users scheduled to a slot earlier in the frame will not be assigned later, since each user is scheduled only once at most. Thus, gathered information of a scheduled broadcast will not effect a later change in the schedule as it can in the combining reception case, where multiple user schedulings are possible. It is seen, therefore, that in the single reception case it is not worthwhile to make a distinction between real time and prior time scheduling, as shown in Table 1.

The last distinction made is that of scheduling sequentially according to the users or according to the slots. This difference is made only in the case of single reception for this

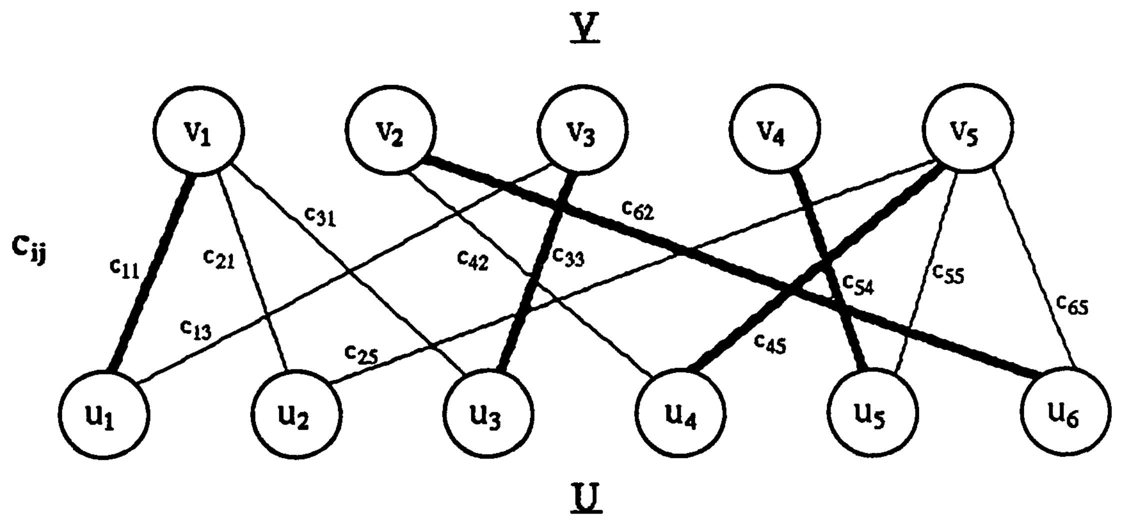


Figure 1. Random Bipartite Graph Matching Example, Q=2.

Table 1. Scheduling Algorithms Considered by Type.

	Real Time	Prior Time
Single Reception	Simple (slot-oriented and user-oriented) Greedy (slot-oriented and user-oriented) Unweighted Maximum Matching	
Combining Reception	Traditional Retransmission: Simple Greedy Incremental Redundancy: Simple Greedy	Traditional Retransmission: Simple Greedy Balanced Matching Incremental Redundancy: Simple Greedy Balanced Matching

thesis, since it will be shown later that using both methods is unnecessary in the combining reception case. As in [4], the slot-oriented method considers each slot individually, scheduling a user by the algorithm in use, and the user-oriented method considers each user sequentially, scheduling the slots.

The algorithms contained within each of the four boxes found in Table 1 may be compared on the basis of their performance. However, one cannot effectively compare performance results of algorithms contained in different boxes, since different assumptions are made concerning performance measures and model implementations. For instance, although the throughput may be comparable for the prior time combining reception algorithms and the real time combining reception algorithms, the two different approaches should not really be compared since they represent different modelings of alternative versions of the scheduling problem. It is clear that a thorough explanation of the various models is needed in addition to descriptions of each of the algorithms. In Chapter II, the single reception algorithms will be discussed, and in Chapter III the combining reception algorithms will be introduced. In Chapter IV, the specific implementations of the algorithms in the simulations will be explained, as well as definitions of the performance criteria used for all the prior time algorithms. Data analysis is performed in Chapter V, and conclusions concerning the performance of these systems is included in Chapter VI.

II. DESCRIPTION OF THE SINGLE RECEPTION ALGORITHMS

Consider the slot-oriented method as explained in Chapter I. In the single reception model, each user is allowed to be scheduled in no more than one slot. Clearly, this modeling assumes that a single reception from a user will be highly reliable. However, since the value of Q may vary to be as large as one requires, each user has many different slots in which it can be scheduled, depending upon the algorithm used. The single slot selected may then be chosen for the greatest possible reliability of reception. Thus, it is possible to achieve highly reliable results when only one reception is used.

There are two different measures used in the construction of schedules, the load and the connectivity. The "load" is the sum total of weights derived for various edges of a graph that can indicate either overall graph performance or the weighted performance of a single slot. The scheduling algorithms use the weights c_{ij} in Figure 1 for this computation. Load is the primary consideration used in the combining reception cases since the results of the multiple receptions are combined. In these algorithms, an overall balance of weights, or load, is desirable. The importance of load will be discussed in the next chapter. However, in the single reception cases, the approach is to try to schedule as many of the users as possible, a result of limiting each user to only one slot. "Connectivity" can then be defined for the slot-oriented method as the number of users scheduled to slots, with the greatest possible connectivity being desirable. The definition of connectivity is similar if one uses the user-oriented approach. In such a case, it is the number of slots scheduled to users. Thus, connectivity is also an important factor in the measure of overall performance of these algorithms. The single reception method stresses the importance of connectivity over the importance of total load on the system.

This is not necessarily the most effective approach for scheduling, though, since it may be possible to schedule fewer users and achieve a better use of the channel for the remaining users. This effect exists since the overall approach used here allows all possible links in the model as visualized by Figure 1 be considered equally. No preference is given to the perceived levels of other-user interference, which is measured by the load. This increases scheduling speed, and it allows connectivity to be the performance measure of the system. The issue of secondary interference is important here, also, since there are more slots than users, but there is only one chance per scheduled user to complete a successful broadcast. Thus, the effects of secondary interference can be studied for the single reception case.

As in Table 1, there are three main types of algorithms to consider. The first two, simple and greedy, appear in all quadrants of Table 1. These two have further distinctions in the form of slot-oriented and user-oriented methods, as explained in Chapter I. The other one is a version of the unweighted maximum matching algorithm for bipartite graphs as found in [5]. The unweighted algorithm produces a schedule of maximum connectivity. The weighted version will not be considered here, as its solutions have the same connectivity of those found using the unweighted version, yet the solutions that are found use the unneeded extra constraint of optimal load.

A. Simple Algorithm

Consider one of the two axes of the graph model for the problem as presented in Figure 1: choose either the slots or the users. The strategy here is to move down the chosen axis in sequential order, scheduling any random, unscheduled node on the opposite axis that is pre-assigned to the current chosen axis node. This algorithm is the least complex to implement and the quickest to run. Its simplicity is the basis for the designation "simple algorithm."

An example of this type of algorithm is illustrated in Figure 2. Here, the chosen axis is the top one, consisting of the slots, so that users will be assigned to them. This is an example of the slot-oriented method. At v_1, u_2 has been randomly chosen among the four users pre-assigned to v_1 to be scheduled. Similarly, u_1 is scheduled for v_2. The choice for v_3 is then between u_3 and u_4 since u_2 has already been scheduled; as shown, u_4 is chosen. Similarly, the choices for v_4 are u_3, u_5 and u_6, not u_1, with u_6 randomly scheduled. Note that at v_5, a user can be scheduled, as in the previous slots. Since within the set of pre-assigned users, there exists at least one user not already assigned, and there is no conflict for the choice to be made for the scheduling. This situation is called a "non-conflict."

A "conflict," then, is one where all pre-assigned nodes have been previously scheduled. In this situation, the chosen axis node cannot be scheduled, as illustrated in Figure 3. Again, the chosen axis is the slot side, and the pre-assignment is the same as in Figure 2. The first four slots are randomly scheduled, but a conflict occurs at v_5. All of the pre-assigned users for this slot, u_5, u_2 and u_6, have been previously scheduled to v_1, v_3 and v_4, respectively. No user can then be assigned to v_5.

These conflicts can often occur during execution of the simple algorithm, especially near the end of the chosen axis. This occurs because of the randomness of the scheduling performed at each node of the chosen axis. Since each scheduling does not take into account the possible effects on previous schedulings, or possible ones later along the axis, some of the scheduling may be inefficient in the sense of connectivity. The chances of such conflicts occurring can be lessened by using a more careful algorithm, such as the following one, the greedy algorithm.

B. Greedy Algorithm

The term "greedy" has been used in the literature to describe an algorithm that works indiscriminately, or with less than optimal care, but uses some pre-determined bias to perform its task. The greedy algorithm used here performs with the randomness of the simple algorithm and with no ability to change earlier decisions to accommodate more optimal schedules. In this sense, it is similar to the simple algorithm, but it is the use of bias in making scheduling choices that differentiates the two.

Again, the strategy is to choose one of the two axes and move down the chosen axis in sequential order of nodes. At each chosen axis node, the algorithm tries to schedule one of the unscheduled opposite axis nodes that is pre-assigned to the current chosen axis node. The choice between nodes not already scheduled is made on the basis of priority in the greedy algorithm. "Priority" is defined here as the number of pre-assigned chosen axis nodes that an opposite axis node has left in which it can possibly be scheduled. Preference for scheduling to a chosen axis node is given to the opposite axis node with the least priority number, with ties decided by random choice, assuming a non-conflict exists. All opposite axis nodes begin with the same priority, and as one loses a scheduling competition for a chosen axis node, its priority is decremented by one. Note that the initial priority for all opposite axis nodes is (Q-1) since each node has Q chosen axis nodes to which it is pre-assigned. The first time each opposite axis node is considered, there are (Q-1) future chosen axis nodes to which it can possibly be scheduled if it loses its first competition. Of course, scheduled nodes are taken out of future consideration in the single reception model, as are nodes that lose a competition when their priority is zero.

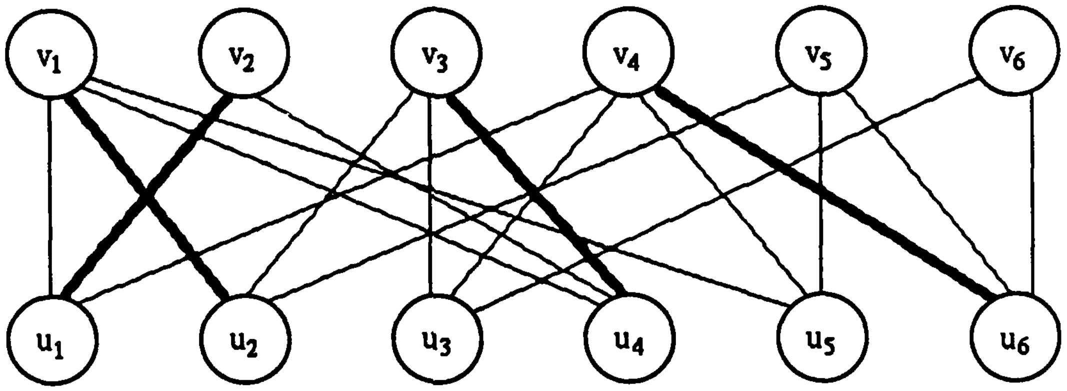


Figure 2. Simple Algorithm, Non-conflict at v_5.

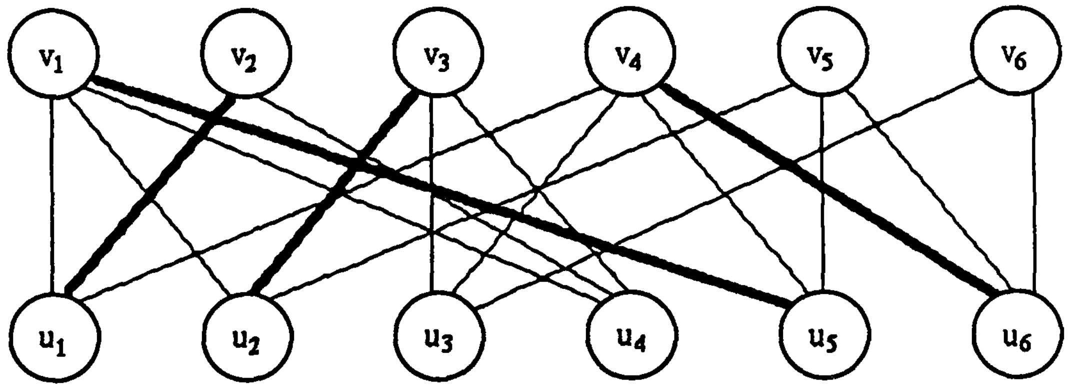


Figure 3. Simple Algorithm, Conflict at v_5.

It is possible to use other measures of priority, and even to decrement priority by non-integer amounts, but the basic method described above allows this algorithm to work with speed comparable to the simple algorithm, yet provide better connectivity. In fact, each chosen axis node is in competition only once, with no more than αQM total opposite axis node considerations, since there are αQM total edges in the graph representing the pre-assignment. Since the same applies for the simple algorithm, the greedy and simple algorithms both require O(QM) computations. Of course, the greedy algorithm may work slightly longer since the schedulings consider all pre-assigned opposite axis nodes, as opposed to the random choices made with the simple algorithm, which may not consider all such nodes.

An example of the greedy algorithm is illustrated in Figure 4. The same pre-assignment as the previous examples is used, and the chosen axis is again the slot side. The algorithm always makes a random choice on the first slot since all users have the same initial priority. In this case, the initial priority is two, since Q=3, and v_1 is scheduled with u_4. Since u_4 has been scheduled, v_2 must choose u_1 since no other pre-assigned user exists for v_2. At v_3, the greedy algorithm must act: u_4 is not considered since it has already been assigned, leaving a choice between u_2 and u_3. The priority for u_3 can be seen to be 2 since this is the first time it is in competition for a slot. The priority for u_2 is 1, however, since it has already lost a competition, namely, that for v_1. The slot is then awarded to u_2 since its priority number is lowest among all unscheduled pre-assigned users. A similar choice exists at v_4. With u_1 out of consideration, note that the priority of u_3 is 1 after losing the competition for v_3, and the priority of u_5 is 1 after losing the competition for v_1. The priority of u_6 is 2, greater than that for u_3 or u_5, so that the random choice must be made between u_3 and u_5.

Note that there will be no conflict anywhere in this graph as currently scheduled, so that all users will be scheduled. As in the situation with Figure 3 and the simple algorithm, though, it is possible that a conflict can appear, but notice how the greedy algorithm tends to avoid such events better than the simple algorithm. This better performance results because a user that has failed to be scheduled previously has more of a chance of being scheduled before it loses all its chances, whereas the simple algorithm gives such a user the same random chance at each slot for which it is in competition. Thus, the greedy algorithm may have more successful schedulings near the end of the axis than the simple algorithm. Of course, this algorithm is still not optimal, since conflicts can often result when α is large, and the algorithm cannot change earlier choices to possibly avoid such conflicts. This type of action to optimize connectivity can be performed by the next algorithm, the maximum matching algorithm.

C. Unweighted Maximum Matching Algorithm (Maxmatch)

The development of the workings of the maximum matching algorithm, which henceforth shall be referred to as "maxmatch," is quite long, involving several proofs and many graph theory definitions. Such a discussion, as well as the maxmatch algorithm, may be found in [5], but a basic description is given here. Consider the graph representation of Figure 1. The set of edges chosen for the schedule is built up one edge at a time, forming, as in the figure, a collection of bold edges. The objective is to add as many bold edges as possible, so that the schedule has as many connections of slots and users as can possibly be found. Using our earlier terminology, this means that the schedule should have the maximum connectivity possible.

Imagine that a partial matching exists, consisting of a set of bold edges. The maxmatch algorithm operates by augmenting this partial schedule by one edge, if possible,

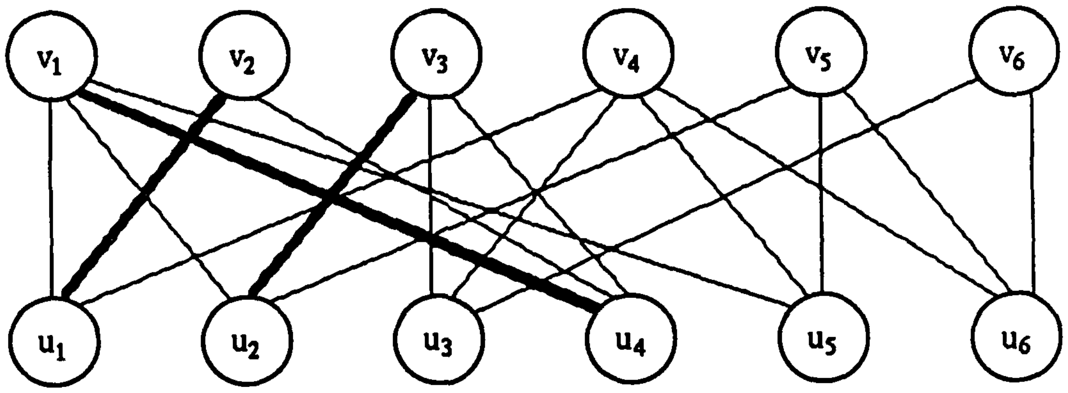


Figure 4. Greedy Algorithm, Illustration of Choices at v_3 and v_4.

so that each pass of the algorithm adds one more connection to the schedule, continuing until no more edges can be added. "Augmenting" is the process of rearranging the past set of edges, if necessary, so that the extra edge can be added to the schedule. This involves finding an alternating connected set of scheduled and non-scheduled edges, or a path, beginning and ending with non-scheduled edges. The scheduled edges in this path are then changed to non-scheduled edges, and vice versa. Since there originally is one more non-scheduled edge than scheduled edges, it is easily seen that this process creates a schedule with one more edge than the previous schedule.

This process is illustrated in Figure 5. Consider the initial scheduling of Figure 5(a). The five edges in this schedule each have been chosen without the need for discovering an augmenting path, but clearly u_2 cannot be added to the schedule simply by adding a free edge, since one does not exist. In this situation, an augmenting path needs to be found if u_2 is to be scheduled. Several exist, but one is illustrated in Figure 5(b). Again, the process by which augmenting paths are found will not be discussed here, but such a description may be found in [5]. As stated earlier, the augmenting path begins and ends with non-scheduled edges, and the path alternates between such edges and scheduled edges. The roles of the scheduled and non-scheduled edges in this path are then reversed, producing the schedule of Figure 5(c), which has one more edge than the initial schedule. This is the maximum matching, and in fact, all users are now scheduled, which was seen to be achievable in the discussion of the greedy algorithm.

Although this algorithm is guaranteed to produce the maximum connectivity, it is also the most time-consuming of the three in this section. As stated in [5], the complexity of this algorithm is $O(M^3)$, although there exists a version that has complexity $O(M^{5/2})$ [6]. Note that there is no distinction between slot-oriented and user-oriented methods for this algorithm. This is due to the fact that the algorithm is guaranteed to produce a schedule of maximum connectivity, so that attempting to schedule from either axis gives the same level of connectivity, and as will be seen later, the same level of average throughput performance is achieved.

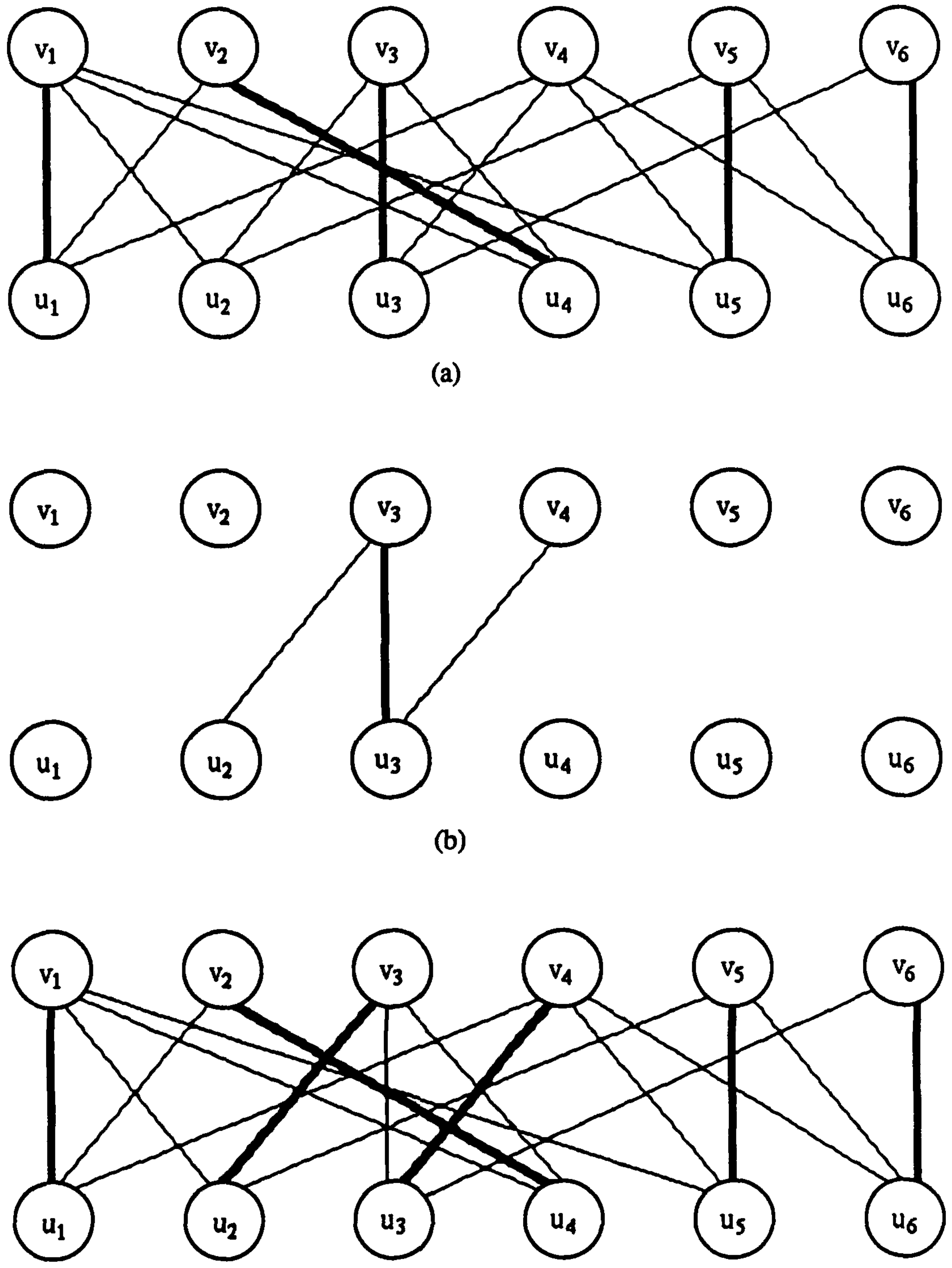


Figure 5. Maxmatch Algorithm. (a) Initial Schedule. (b) Augmenting Path. (c) Final Schedule.

III. DESCRIPTION OF THE COMBINING RECEPTION ALGORITHMS

Consider again the slot-oriented method presented in Chapter I. The combining reception model allows each user to be scheduled to multiple slots. Redundant information may be received for processing, allowing possibly better performance than for the single reception model, especially when a single reception may not be highly reliable as assumed for the single reception model. The conflict situations that were possible in the single reception case occurred when a slot could not be assigned since all its pre-assigned users had previously been scheduled. By allowing multiple schedulings for the users, these conflicts never occur. With the combining reception model, the slot-oriented method allows all slots with any pre-assigned users to be scheduled. Thus, a maximum connectivity is always achieved. For this reason, the slot-oriented method will be the only one considered. Thus, connectivity, the important measure of the single reception model, is not as important as the overall load of the schedule achieved. Load, as explained earlier, involves the overall balance of weights assigned to the edges of the graph visualization of the problem, or the c_{ij} of Figure 1. These are used in the combining reception case since the results of multiple receptions are combined, making an overall balance of weights, or load, desirable. Secondary interference effects are not as crucial here, since every user has the possibility of getting a complete transmission through in several slots, thus cancelling out the effects of the interference in earlier unsuccessful slots.

There are three algorithms considered here, as found in Table 1. The first two, simple and greedy, are very similar to the versions presented earlier for the single reception model, and because of their simple nature, do not involve load balancing. That task is performed by the third algorithm, known as the balancing algorithm. As before, the simple and greedy algorithms are the quickest, easiest to implement and least optimal, and the balancing algorithm is the most optimal, most complex and slowest to operate of the three. The first two algorithms also have both real time and prior time applications; a discussion of the prior time versions follows, with a separate discussion of the real time versions appearing later. The balancing algorithm is designed to produce a near-optimal scheduling performance. Other, more thorough algorithms exist that can provide better, or even optimal results, but the balancing algorithm will provide satisfactory results for this study.

A. Prior Time Algorithms

The prior time versions of these algorithms all operate in a manner similar to that of the single reception case, where the performance of any broadcast earlier in the frame does not change the scheduling of any slot later in the frame. There is also the possibility of using incremental redundancy techniques here, but the algorithms using the traditional retransmission method are presented first.

1. Traditional retransmission

The traditional retransmission method, as stated in Chapter I, uses scheduled further broadcasts of a given user to possibly recover an originally faulty broadcast by relying on the possibility that these other broadcasts will be of better quality. The success of this method depends on the other-user interference levels in the slots in which the user's packet will appear in a frame, as well as the number of times that the user has its packet scheduled. For instance, if a user is only scheduled once in the frame, then a faulty broadcast will cause packet loss. However, if the user is scheduled twice, there is a possibility that the second broadcast will be correctly received. The probability of this being possible is decreased if there are more users transmitting in the second slot than in the first, but then the other factors of secondary interference come into play here, namely, the error control coding and the spread-spectrum frequency hopping patterns.

a. Simple algorithm

The strategy of this algorithm is nearly the same as that for the single reception case. The algorithm moves through the slots in sequential order, scheduling any random user that is pre-assigned to the current slot. Note the difference between this strategy and that of the single reception model: the restriction that each user be scheduled in only one slot is dropped here. This allows any slot with pre-assigned users to be scheduled, and it also allows a user to be scheduled to several slots. A schedule constructed using this algorithm is illustrated in Figure 6. Note that two users, u_2 and u_4, have been scheduled in two slots, and two other users, u_1 and u_5, will not have their broadcasts heard in any slot. This imbalanced situation is a risk that can occur when using the simple algorithm due to the randomness of the choices made.

b. Greedy algorithm

This algorithm is also nearly the same as its single reception counterpart. As with the simple algorithm described above, the only difference is the removal of the restriction that the user be unscheduled. Thus, the competition at a slot consists of all pre-assigned users, with scheduling of users occurring by comparing their priority numbers, a measure of a user's performance in past competitions. This again allows any slot with pre-assigned users to be scheduled, and it also allows a user to be scheduled to several slots.

A modification is needed for the algorithm to be complete. As before, when a user loses the competition for a slot, its priority number is decremented by one. The issue to be addressed that was not needed in the single reception model is what to do with the priority number when the user wins the scheduling competition. This needs to be considered in this model since the decrementing is used to indicate a measure of past failure for future competitions, but some measure of past successes is not necessary for the single reception model. Several possible actions on the priority number were investigated, but the one selected works as follows: if a user wins scheduling to a slot, nothing is done to its priority number. Incrementing the priority number was an action also attempted in simulations, but it was observed that this has the same effect as not doing anything, which is a less complicated action. Decrementing the number is undesirable, of course, since it would then provide the same effect as losing a competition.

This algorithm is demonstrated in Figure 7. The choice for v_1 is, as usual, random, with u_4 being scheduled. At v_2, however, note the priority distinction. Since it lost competition for v_1, u_1 has priority 1, while u_4 has priority 2 since it won the competition for v_1, and thus its priority was not decremented. Thus, u_1 is scheduled to v_2. Note that at v_3, u_2 has priority 1 for losing at v_1, u_3 has priority 2 for being considered for the first time, and u_4 has priority 1 for losing at v_2. A random choice must then made between u_2 and u_4, with u_4 winning as shown. Similar random choices are made at v_4 and v_5 between users of equal priority, with u_3 being scheduled to v_4, and u_2 being scheduled to v_5. Finally, u_6 easily wins scheduling to v_6. Again, as with the simple algorithm, note how a user that has won more than one slot, u_4, can leave a user with no scheduled slot. In this case, u_5 cannot get its broadcast through.

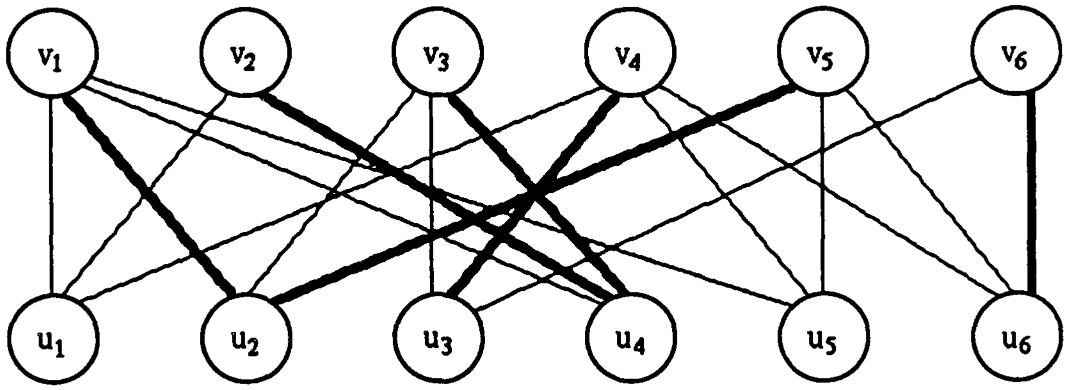


Figure 6. Simple Algorithm, Combining Reception.

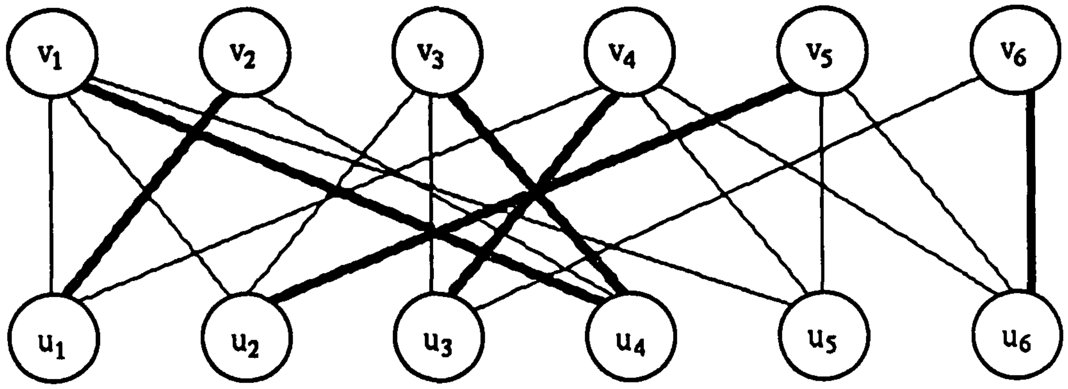


Figure 7. Greedy Algorithm, Combining Reception.

c. Balancing algorithm

The balancing algorithm requires a longer discussion than the previous two. First note in Figure 8 that several new variables require definition. This algorithm deals with the balancing of user "levels," so that situations such as that in Figure 6 occur less often. It tries to equalize a function of the weights c_{ij} for each user u_i, with this function being called the "level." In this case, the function used is the sum of all c_{ij} such that user u_i is scheduled to slot v_j. In other words, if the level for user u_i is denoted by d_i, then,

$$d_i = \sum_{j^*} c_{ij^*} \tag{1}$$

where the sum is over all j* such that v_{j*} is scheduled to user u_i. The values for each of the d_i are indicated in Figure 8. Note that d_5 is zero since u_5 is not scheduled, and d_4 is $(c_{41}+c_{43})$ since it is scheduled to both v_1 and v_3.

With the c_{ij} chosen carefully, the balancing of the levels can improve schedule performance in several ways. First, just balancing the levels can allow more users to be scheduled, since a large level can represent the scheduling of user to several slots, which as noted before, can mean that other users will not be scheduled, so that they have small or zero d_i. An algorithm which produces a schedule with more equalized d_i will then allow users not scheduled to be scheduled more often, and the users scheduled often to be scheduled less often, with the overall effect being that more users are scheduled. Next, the c_{ij} may be chosen to be a function of the number of pre-assigned users at slot v_j. This choice of weight assignment will be addressed later, but the overall effect is to favor slots with fewer pre-assigned users, if possible. This, in combination with the above effect, allows the d_i to consist of a sum of less discrepant c_{ij}, which means that there will be less waste when a user is scheduled more than once, since before balancing it can be possible to have very high reliability in one slot and relatively poor reliability in another. There is then less of a reason to broadcast in the latter slot, whereas another user may be able to utilize this slot for its broadcast, which can be achieved by balancing. Clearly if one is interested in high overall average performance of the system, rather than the superior performance of only a few user broadcasts, level balancing is very desirable. This heuristic described above, which is based on swaps of user-slot schedulings to improve balance, produces the desired overall performance.

The algorithm begins with an initial schedule, as indicated in Figure 8. The b_j variable represents the user to which a given slot v_j is scheduled, if it has pre-assigned users. Note that in Figure 8 all the b_j have user values since all the v_j are scheduled. A slot v_s is then chosen at random, and if it has more than one pre-assigned user, then a search begins among such users to find a user u_r with the smallest level value. After such a user is found, the following test is performed:

$$d_r \leq d_{b_s} - c_{b_s s} \tag{2}$$

If true, then v_s should be rescheduled to u_r. The reasons for using this test will be explained shortly. The level values for the affected users are altered in the following way:

$$d_{b_s} \leftarrow d_{b_s} - c_{b_s s} \tag{3}$$

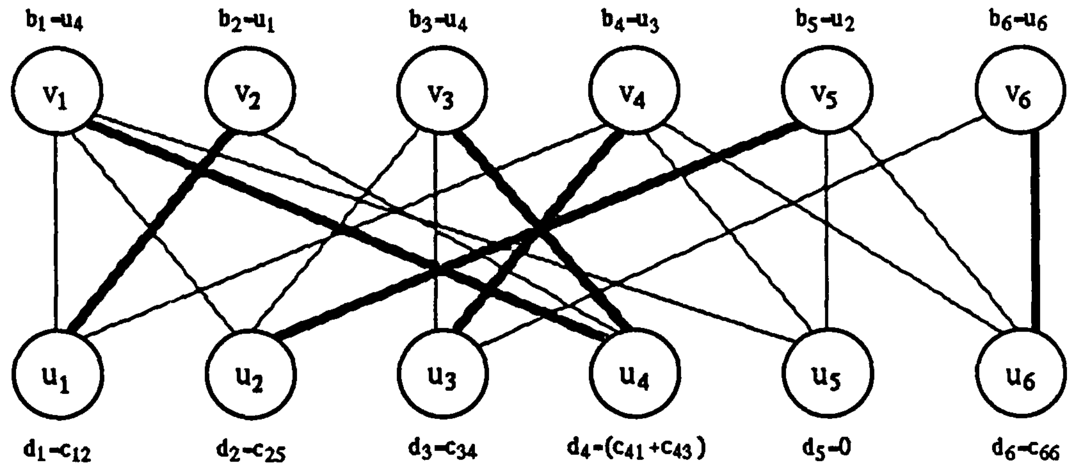


Figure 8. Balancing Algorithm, Illustration of Levels.

$$d_r \leftarrow d_r + c_{b_s s} \qquad (4)$$

These are the new values since what was previously b_s is no longer scheduled to v_s, so that $c_{b_s s}$ is no longer in the set of weights that make up d_{b_s}. Similarly, u_r is now scheduled to v_s, so that $c_{b_s s}$ is now in the set of weights that make up d_r.

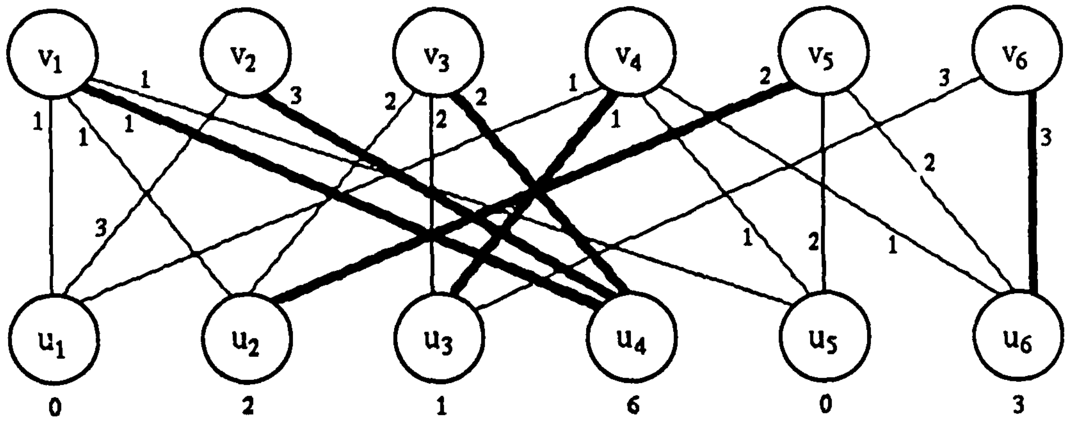
From Equations (3) and (4), it is now seen why Equation (2) was used to test if a rescheduling was necessary. If a rescheduling is to result in a more balanced situation, then the level for u_r should be less than the reduced level for u_{b_s}, since then the discrepancy between the levels d_r and d_{b_s} will be smaller, which shows greater balance than before. If, instead, equality is achieved in (2), then the rescheduling will not hurt the degree of balance, since the discrepancy between the levels d_r and d_{b_s} will be the same. A timer variable must also be incremented, as the algorithm stops trying to balance when better balance cannot be obtained. Since one pass of the algorithm chooses a slot at random, in order to guarantee that as many slots as possible, if not all, are considered before timing out, this variable must reach a value equal to several times the number of slots. Conversely, when the test strictly satisfies the inequality of (2), this timer variable is reset.

An example of this algorithm is demonstrated in Figure 9. In Figure 9(a), the familiar pre-assignment is presented with an initial schedule, a set of weights and the corresponding levels. The numbers used for the levels are not the same as those used in the actual simulation, but they will serve for demonstration purposes, as well as provide for greater simplicity in viewing the algorithm. Here, though, as in the simulation, the weights have been assigned in the following fashion: all edges emitting from the same slot have the same weight, and these weights are a decreasing function of the "degree" of a slot, which is defined as the number of edges emitting from that slot. In particular, the weights for a slot's edges are computed by finding the maximum degree among all the slots, adding one to this number, and then assigning weights by subtracting the slot's degree from this number. Note the large discrepancy between the level of u_4 and those of u_1 and u_5. Thus, this schedule could clearly benefit from balancing.

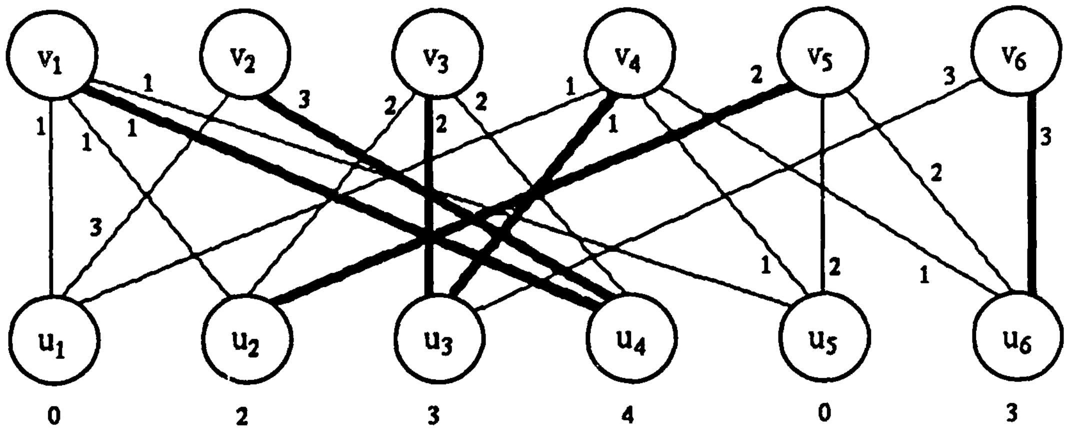
The first step is to choose a random slot; v_3 is selected. Next, the pre-assigned user with the smallest level is found to be u_3 among u_2, u_3 and u_4. The test of Equation (2) is then applied, with $d_3=1$, $b_3=u_4$, $d_4=6$, and $c_{43}=2$. Thus, since $1<(6-2)=4$, balancing can be performed by rescheduling. From Equations (3) and (4), the new levels are found to be $d_4=(6-2)=4$ and $d_3=(1+2)=3$, and the rescheduling gives $b_3=u_3$. This is the situation illustrated in Figure 9(b).

Further balancing can be performed to improve the schedule. If the next slot chosen is v_1, then it is easily seen that v_1 can be rescheduled from u_4 to u_1 or u_5. Choosing u_1 for now, the new levels are $d_1=1$ and $d_4=3$. Finally, if the next slot chosen is v_4, then v_4 can be rescheduled from u_3 to u_5, resulting in new levels $d_3=2$ and $d_5=1$. No further rescheduling can occur, since any attempt will result in failure for the test of Equation (2). The algorithm will time out with this final schedule, illustrated in Figure 9(c).

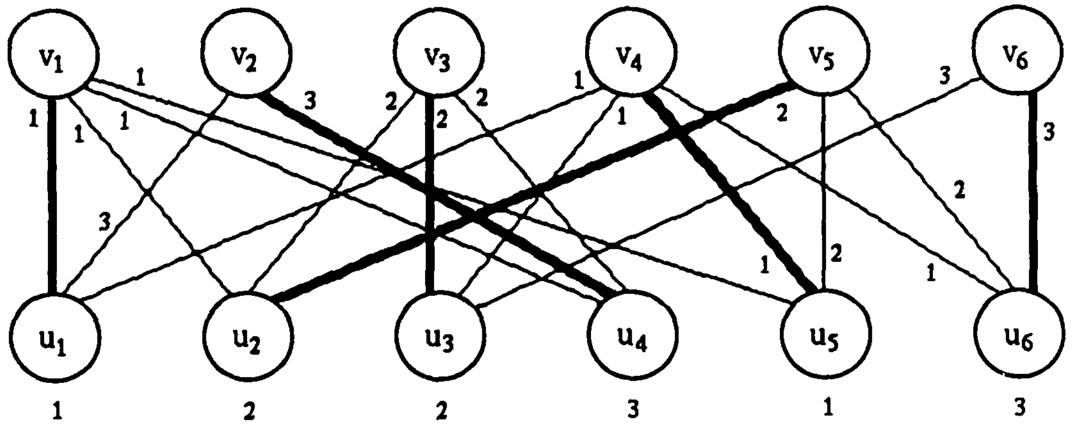
Note that in all three of the previous schedules the average level is the same. This phenomenon thus shows the need for a different measure of balance. Although the mean is the same for the three schedules, the variance of the distribution of the levels in these schedules differs. In fact, note that the variance of the levels decreases from its largest value in Figure 9(a) to its lowest value in Figure 9(c) as the level distribution becomes more centered about the mean level. Thus, an improvement in balance can be measured by observing a decrease in the variance of the distribution of the user levels. This is an



(a)



(b)



(c)

Figure 9. Balancing Algorithm. (a) Example with Levels. (b) Example After Balancing at v_3. (c) Final Schedule After Balancing at v_1, Then v_4.

important measure of the algorithm's performance. Again, note the schedule of Figure 9(c). This schedule, where all slots are scheduled and no user is scheduled more than once, is a special situation which will always result in eventual termination of the algorithm with that schedule or one of identical performance, since all remaining attempts will either result in equality or failure in Equation (2).

2. Incremental redundancy

The discussion in Chapter I explained in brief the method of incremental redundancy, and in light of that discussion, it is easy to see how incremental redundancy functions here. Incremental redundancy [3] involves coding the Q transmissions of a user's packet within a frame as a larger super-packet, and then decoding by using the parts of the sub-packets that are successfully received. Although it can be applied, the method does not really affect the true workings of the prior time algorithms since the nature of prior time is that the information gathered in the frame on earlier broadcasts does not change the later schedulings, as it may in the real time case. However, use of incremental redundancy does improve the performance and it requires a different performance analysis. This means that incremental redundancy does not change the prior time algorithms, but it does change their performance from that of traditional retransmission, and for this reason, both methods are indicated in Table 1. In actual implementation, the scheduling algorithm would not change, but the decoding algorithm used would differ, causing the change in performance.

It is worth noting that incremental redundancy has an especially effective extension to the balancing algorithm. In this algorithm, the stated desire is to provide users with the highest average number of schedulings possible, given the non-optimality of the heuristic. The use of incremental redundancy then allows the algorithm to balance the number of successfully received symbols. The simple and greedy algorithms can use incremental redundancy to improve performance, but their methods are not attempting to schedule users as often as possible in the way that the balancing algorithm does. The balancing method then couples well with incremental redundancy in prior time since this model assumes that additional schedulings must occur. Therefore, if one is using incremental redundancy, balancing is desirable. These additional broadcasts are utilized in a way which is more efficient than that which occurs with traditional retransmission.

B. Real Time Algorithms

If processing power and speed do not come at a premium, the use of scheduling algorithms in real time can improve performance. The real time algorithms are similar to the prior time versions in many ways. First, though, note that the balancing algorithm is no longer considered in the real time case, since this algorithm takes a graph-wide approach to producing a schedule, like the maxmatch algorithm, and thus cannot be implemented in real time as defined.

Real time algorithms, as described in Chapter I, primarily differ from their prior time counterparts in that they are able to adapt later schedulings to performance of earlier broadcasts. This allows for increased performance over the prior time case. The corresponding prior time algorithms are thus a special case of the real time algorithms. To operate, the real time algorithms move down the frame, in sequential order or slots, scheduling users to slots according to some pre-ordained strategy of competition. The two strategies used here are, as in Table 1, the simple and greedy algorithms. They also have corresponding cases of both traditional retransmission and incremental redundancy.

Throughput analysis for the real time algorithms is not performed in this thesis, but a brief discussion of these algorithms is presented in this section. Some of the conclusions found in the prior time case can apply to real time, but the actual simulation of these algorithms is not part of this thesis.

1. Traditional retransmission

The algorithms in real time become decision strategies rather than the complete graph methods as used in prior time. This occurs since the "algorithm" acts on only one slot at a time rather the entire graph. Of course, for the simple algorithm, this action is nearly identical to that in the prior time case, but the greedy algorithm does still use some graph-wide information in making its decisions, namely, that which is gained from the slots scheduled earlier in the frame.

An important difference for both these algorithms is in the treatment of successful user transmissions. Since it is possible for multiple slots to be assigned to a single user, additional broadcasts do not help a user's transmission if the original transmission was successful, assuming error-free reception. Thus, the traditional retransmission algorithms in real time can disregard a user in later competitions in favor of previously unscheduled or faulty user broadcasts. This deviates from the prior time algorithms in the main sense that no assumptions were made on the success of early slot broadcasts since no broadcasts have occurred in the prior time. Of course, this means that once a packet is successful, the real time algorithms treat users in the same way that the single reception algorithms did in the sense that later possible schedulings are disregarded after a successful scheduling. Again, the prior time greedy algorithm takes into account earlier schedulings, but not earlier transmission successes. As in the prior time case, though, an entire packet rebroadcast is needed for faulty packet recovery using the traditional retransmission method.

a. Simple algorithm

The simple algorithm moves through the slots in sequential order, scheduling any random user that is both pre-assigned to the current slot and which has not had a previously successful broadcast. This can be seen to be the strategy of the prior time case with the additional constraint that the users in competition have not successfully transmitted, which means that either the user has not previously been scheduled or the user was scheduled but all the previous schedulings provided faulty transmissions. This additional constraint has the effect of removing the guarantee that any slot with pre-assigned users will be scheduled since it is possible that all the users in competition for a slot have been scheduled and have had successful previous transmissions. However, the constraint keeps the effect of allowing a user to be scheduled to several slots, although it is likely that if the probability of packet success is high on the average, the number of times that a user will be scheduled will be lower than that for the prior time case since that user can be removed from later competition if necessary. This latter effect also changes the imbalanced situation of the prior time case where some users can be scheduled several times and others may not be scheduled due to the randomness of the choices made. Now, since the number of schedulings a single user receives will probably decrease, more users should have a chance of being scheduled, especially later in the frame after successful broadcasts have occurred.

b. Greedy algorithm

As with the simple algorithm above, the change from the prior time case is the addition of the constraint of scheduling users who have not had a previously successful broadcast. The effect on this algorithm is slightly different, though. In order to implement this constraint, an additional modification to the prior time algorithm is needed. As before, when a user loses competition for a slot, its priority is decremented by one, with all users starting with priority (Q-1). However, if a user wins competition to a slot, nothing is done to its priority number only if the transmission is unsuccessful. If the transmission is successful, the user is removed from further competition, as in the single reception case. This change produces the same effects as with the simple algorithm above. More users have a chance of being scheduled due to the probable lower number of schedulings a single user will receive. Note that this effect was produced by switching to the greedy method from the simple method anyway, so that the performance can actually improve further over the simple case if the packet success probability is high. Again, though, since an entire packet rebroadcast is needed for faulty packet recovery using the traditional retransmission method, further improvements in performance must come from using the incremental redundancy method.

2. Incremental redundancy

It may be possible for one to implement incremental redundancy in real time. Again, the scheduling algorithm in the actual implementation would not necessarily change, but the decoding algorithm differs, providing the difference in performance. This change can be thought of as a gain in performance without the use of significant changes in the original scheduling algorithm. Its implementation in real time, however, may be difficult, since the new coding and decoding algorithms will be far more complex. If additional processing power allows incremental redundancy to be implemented in real time, though, the main scheduling algorithm will not suffer significant changes.

There is, however, a change in the decision method in determining the set of users that will be considered for competition for a slot. In traditional retransmission, a user is removed from competition for further slots if its entire last broadcast was correctly received. Since the incremental redundancy technique uses only the successful symbols of a packet, it is possible to construct a correct packet from the successful symbols across several transmissions, eliminating the need for the entire last broadcast to be successfully received. Using this scheme, a user is removed from competition for further slots if the previous broadcasts have combined to provide a successful packet. In the simple algorithm, then, there is no change in the scheduling algorithm, but the change occurs in deciding whether or not a user still needs to be in competition for slots.

The greedy algorithm, however, has a more subtle change. Its consideration of which users continue on for competition for slots is similar to the change shown with the simple algorithm, but the system of weighting the various users that are in competition also changes. This algorithm uses a statistic that considers the past performance of users to determine the selected user. In the prior time case, this was simply an integer statistic that changed whenever a user was unsuccessful in competition and stayed the same whenever a user was successful in competition but unsuccessful in transmission. Integers were used since the unit of consideration was the transmitted packet. The incremental redundancy case, however, allows one to use fractions of a packet since any successful symbol can now be considered. This creates the need for a non-integer statistic that can account for the record of previous competitions and the performance of all of the past scheduled transmissions in terms of the symbols received. Of course, this statistic is also needed in the simple algorithm to determine whether a packet can continue in competition, but it is not used in the selection process of users for a slot.

Part of this statistic is the conditional probability that the current user transmission will be successful given that the previous ones were not. This statistic takes into account the record of past competitions in that the probability will reflect the number of schedulings that were required to get to the current number of symbols required for correct reception. It then also accounts for the performance of previous transmissions in that the probability will be higher if previous transmissions were highly successful. This means that the statistic should increase as more schedulings occur, assuming that the user is not taken out of competition.

The user chosen for scheduling to the slot under consideration should also have fewer slots than other competing users to which it can be scheduled in the future, as in the prior time discussion of the greedy algorithm. Thus, the old weighting system and the above statistic are combined into a new statistic, namely, the product of the number of possible future slots for consideration and the conditional probability of packet success. This is called the "modified priority number," and the user in competition with the lowest modified priority number wins scheduling to the slot. Both components of this new priority are separately computed, so that the number of future slots is decremented by one if a user loses the competition, and unaltered if the user wins competition but has still not completed a successful transmission using the incremental redundancy of all of its previous broadcasts. The computation of the conditional probability, however, involves a greater effort, and will not be part of this thesis.

IV. PERFORMANCE CRITERIA

Each of the previous algorithms was implemented on a computer, but there are many details that need to be described to understand how the final performance measures were produced. Some of these details have already been described in the discussions of the algorithms, but there are several concepts needing clarification in order to present the results of the simulations. Throughput is computed only for the prior time algorithms; therefore, all analyzed algorithms will be assumed to be prior time.

All simulations used a standard value of 100 time slots per frame, or $|V|=100$. The number of users, denoted by $|U|$, was varied between 10 and 100 to increase the size of the system. Note that the number of users never exceeded the number of slots. The explanation for this is that for all values up to 100 users, the reasons why a user may not be scheduled are because of the competition for slots, which depends on the pre-assignment, and because of scheduling inefficiency, which depends on the algorithm selected. Beyond 100 users, this competition for slots also causes a steadily decreasing fraction of the users to be scheduled if the number of users is increased. This reason is independent of the algorithm used, thus the number of users did not exceed 100. The term α introduced in Chapter I is now seen to be $|U| / |V|$; α then varies between 0.1 and 1.0. The rebroadcast redundancy Q was varied between 1 and 4. Higher values were not used, as the observed performance effects of increasing Q beyond 4 were negligible.

For a given algorithm, the value of Q, and the value of α, the average throughput of 100 trials is presented. This involved producing 100 different random graphs of the form presented in Figure 1, constructing a schedule for each graph, and then combining the results of the 100 schedules in an appropriate manner depending on the measure of performance. Each graph was represented in the computer by either a linked list, which consists of a related set of smaller single dimension arrays, or by a weighted incidence matrix, which is a larger two-dimensional array. The slots and users were connected by edges at random using identical random number generators contained within each program, forming a pre-assigned graph that conforms to the values of α and Q desired. Since the random number generators were internal and controlled, it was possible to create the graphs in the same order for each algorithm. Therefore, for each value of α and Q, the same 100 graphs have been considered for each algorithm. This makes comparison of algorithm performance more convenient and much more valid.

Each packet was coded digitally for the channel using a (32,24) Reed-Solomon code and transmitted Q times within a frame. Each packet transmission was one slot length in time. A frequency-hopping, spread-spectrum, synchronous signaling system was also used for the transmissions, and for the purposes of this simulation, 32 distinct hopping frequencies were used.

A. Throughput with Traditional Retransmission

This throughput measure was used in all the single reception cases and in the combining reception cases that used traditional retransmission. For each digital transmission to a slot from a given user u_i, there was a probabilistic assumption of the expected number of packet symbols that would be successfully received. Normalized by the total number of symbols transmitted, this number becomes the expected fraction of successfully received symbols per slot, or the expected number of successful slots. This number can be computed from the following formula, as presented in [7]:

$$c_{ij} = \left[1.0 - \left[1.0/q\right]\right]^{k_j - 1} = P_{ik_j} \tag{5}$$

The term q represents the number of hopping frequencies used. Thus q=32, and (1.0 / q) is the probability of a hit in a transmission's dwell interval for the synchronous hopping system used, as in [7]. The number k_j, defined as the degree of the slot v_j, is the total number of users transmitting in slot v_j. Each edge connected to this slot has weight P_{ik_j} as shown in Equation (5). Thus, as presented in the discussion of the balancing algorithm, Equation (5) is a decreasing function of the slot degree, as desired, since the greater the other-user interference, the less the expected number of successfully received symbols for the scheduled transmission. An additional degradation factor, d, can be introduced here to represent greater background noise by multiplying c_{ij} by a factor less than one. This makes packet reception more sensitive to additional interference caused by other users, producing a less ideal channel. For instance, multiplying by d=0.9 will reduce the number of symbols that are correctly received, and the overall throughput will be lowered.

By now defining the level of a user as the expected number of slots for a given user, a mean level can be computed. By using Equation (1) to compute the level for each user, and then averaging over all the users, the mean level is found. This assumed a distribution for the values of the levels, as presented in the discussion of the balancing algorithm. Since the P_{ik_j} values for a given slot are dependent on the entire pre-assignment, the user level is conditionally dependent on the slots used for the computation found in Equation (1). Thus, the variance that can be found is the average conditional variance of the levels.

This can be computed by noting that the "level" contributed by a given scheduled slot is a Bernoulli random variable, since P_{ik_j} is given as a success probability of a given trial, or transmission. This random variable thus has mean P_{ik_j} and variance $(P_{ik_j} \times (1.0 - P_{ik_j}))$. These values are summed over the scheduled slots for a given user, which is given as the set of j* in Equation (1). These numbers are then averaged over all the users to produce the variance of the level, which was described as an important measure of balance during the discussion of the balancing algorithm. Note that although this was described as useful for the combining reception model, the concepts of level, mean level and level variance apply just as equally to the single reception model as performance measures. This occurs since the levels can still be computed, and if one averages these values, the users that are not scheduled contribute in a detrimental way to the mean, while those users that are scheduled contribute only one slot's "level" to the average. These contributions imply that the important single reception measure of connectivity factors into the mean level and level variance computations. By reporting the level performance, then, the connectivity is still considered.

Even though the level variance measure is useful, it still has its faults as an overall performance measure. The mean level value provided a useful measure of the average expected successful symbols, but this value is the same for all the combining reception algorithms. In fact, in the combining reception algorithms, for values of Q=2 or greater, the mean level can be greater than one, discounting its value as a probability measure. The variance was shown to be more useful in measuring the balance for the algorithms, but note that it is simply a mathematical value; it does not provide any information on the number of successful symbols, as the mean level did. A more useful performance measure would be the average probability of packet success within the frame, known also as the "throughput."

Since each value of P_{ik_j} is a symbol success probability, the use of the Reed-Solomon code provides a way to find a packet error probability for the slot scheduled. This equation is found in [7], and slight modifications give the following desired formula:

$$G_{ij} = \sum_{l=N-K}^{N} \binom{N}{l} \left[1.0 - P_{ik_j}\right]^{l} \left[P_{ik_j}\right]^{N-l} \qquad (6)$$

This represents a simple binomial sum over the dimensions of the Reed-Solomon code using the symbol success probability for the binomial exponential parameter. Since the same packet is transmitted independently by user u_i in all Q pre-assigned slots, the product of the G_{ij*} for the slots j* to which user u_i is scheduled is the user's frame packet error probability. By convention, if a user is not scheduled, the frame packet error probability is one, since all packet symbols will not be received. From the frame packet error probability, the frame packet success probability is easily found, so that by averaging over all the users, the average frame packet success probability, or throughput, is computed. A formula for this throughput can be expressed as follows:

$$T = \frac{1}{\alpha M} \sum_{i=1}^{\alpha M} \left[1.0 - \left[\prod_{j^* \text{ for user } i} G_{ij^*} \right] \right] \qquad (7)$$

The throughput measure, T, is thus the most useful of all discussed, and it is the only one reported in the data. Note that T will generally decrease with increasing α, and it will generally increase with increasing Q, since the connectivity increases as more edges are added to the pre-assignment. This increase of T must reach a limit, however, since with more edges in the graph comes larger other-user interference, and thus the values of G_{ij*} will increase.

A performance method not considered in this thesis is found in [8], where the single reception model is used, but fixed tolerances are applied to the other-user interference levels. In other words, if the number of users broadcasting in a given slot exceeds a certain fixed tolerance value, the scheduled user's packet is assumed to be lost. The tolerances used vary between zero, representing the case where no other broadcasts can be tolerated in the slot, to the total number of users broadcasting in the frame, αM.

B. Throughput with Incremental Redundancy

The incremental redundancy case requires a different method than the one described above. Since the probability of error for a given transmission now depends on the number of correctly received symbols, there is a need for the probability distribution of correct symbols for each user transmission. This distribution is binomial if the probability is conditioned on the other-user interference present in the slot. The random variables that create this distribution for each user and slot are conditionally independent, then, since interference is caused only by the random hopping patterns of the various transmissions, which produce frequency hits, or by the presence of white noise. If the conditioning is not performed, the distribution is not binomial.

The binomial distribution is then approximated by the Gaussian density function since we are using many users and slots, as the central limit theorem requires. Since the distributions are conditionally independent, the sum of the Gaussian distributions is also Gaussian, so that it is only necessary to keep a running total of the mean and variance of the levels for each user. The expected number of successful slots is still used, computed in (5), to give the mean level P_{ik_j} and the conditional variance ($P_{ik_j} \times (1.0 - P_{ik_j})$) for each transmission of user u_i to slot v_j. These values are summed over the set of slots v_{j*} for each user to give the mean and conditional variance of the number of correctly received symbols for each user:

$$NW_i = N \sum_{j^* \text{ for user } i} P_{ik_{j*}} \tag{8}$$

$$NZ_i = N \sum_{j^* \text{ for user } i} \left(P_{ik_{j*}}\right)\left(1.0 - P_{ik_{j*}}\right) \tag{9}$$

W_i is the mean level and Z_i is the conditional variance of the level. The corresponding number of successful symbols received can then be described as NW_i, where N is the number of code symbols in the Reed-Solomon code. The number of needed successful symbols is K and the variance is NZ_i. The throughput can then be described by the following formula:

$$T = \frac{1}{\alpha M} \sum_{i=1}^{\alpha M} Q\left(\frac{[K - NW_i]}{\sqrt{NZ_i}}\right) \tag{10}$$

where Q( ) is the Gaussian tail probability defined by

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^{\infty} e^{-\frac{t^2}{2}} dt \tag{11}$$

These values are then averaged over the hundred trials to obtain the overall throughput. Note that the computation is simplified through the use of the Gaussian probability, although its approximation is of high quality due to the large number of users and slots considered.

C. Loss Due to Primary Interference

The total rejection rate due to primary and secondary interferences is one minus the throughput, and reflects the total fraction of packets that were not received due to non-scheduled users and imperfect transmissions from those that were scheduled. The primary interference can be measured by finding the average fraction of users whose Q packets were not scheduled. This is the number of users that are rejected due to competition for the slots. If this number is already significant, the primary interference should be lowered to achieve better performance rather than the secondary interference, since the successful packets will always be of high quality. Similarly, if the secondary interference rate is already high in comparison to the primary interference, one must concentrate on lowering the secondary interference in order to achieve better performance, since one may be able schedule more packets, but these will eventually be lost at a high rate due to the secondary interference. Note that the total rejection rate will then always be greater than the primary interference rate, since the total rejection rate reflects the combination of both the primary and secondary interference.

Thus, it makes sense to study the relative error rates due to these two sources of interference in the system in order to make additional judgements on the performance of the algorithms. The tables of these results are presented for analysis in the next chapter.

V. DATA ANALYSIS

All results presented are in the form of graphs or tables that display average user throughput over 100 trials versus the system loading, presented as α (alpha), the ratio of users to slots in the system. The throughputs are computed using the formulas introduced in Chapter IV. The systems are also presented using different values of the rebroadcast redundancy, Q. The throughputs of these algorithms are compared for given values of Q and α. The comparisons are separated according to the divisions in Table 1, so that the single reception algorithms will be presented first, followed by the various combining reception cases illustrated.

A. Performance of Single Reception Systems

The graphs illustrating the performance of the single reception algorithms are presented in Figures 10 and 11. The graphs present the performance of five algorithms: simple and greedy, both with slot-oriented and user-oriented distinctions, and maxmatch. Figures 10 and 11 each show four graphs, each of which utilize a different value of Q.

Figure 10 has d=1.0, or no degradation. The first thing to note is the equivalence of the performance of all five algorithms for Q=1 in Figure 10(a). This occurs because each user only broadcasts once. Any slot with a pre-assigned user is scheduled, with no alternative scheduling for any user, allowing the random scheduling of the simple algorithm to give the same connectivity as the maxmatch algorithm. Also note that the performance degrades quickly as α increases, since more users degrade the system significantly when there are no scheduling alternatives available to the users.

Since alternatives are available for the users for higher values of Q, the performance differs in the other three graphs. The two slot-oriented cases are nearly equivalent along the whole range of α for each value of Q. The user-oriented cases perform better than the slot-oriented cases, but the greedy algorithm consistently outperforms the simple algorithm, as expected. A possible explanation for the better performance of the user-oriented algorithms is the following. Recall that the slot-oriented method affords a view of the users pre-assigned to a slot when scheduling, while the user-oriented method allows a user the view of all its pre-assigned slots. Thus, when there are more slots than users, the user-oriented method allows more scheduling options to be viewed at each step. The increased number of scheduling options then allows these algorithms to perform better. This will not be as great a factor for the combining reception algorithms, since more than one reception per user is allowed. The user-weighting of the greedy algorithm benefits further from this larger view of the graph. Thus, the greedy algorithm naturally outperforms the more random simple algorithm. The smaller view encountered by the slot-oriented algorithms does not give the greedy algorithm any advantage over the simple algorithm.

As expected, the maxmatch algorithm generally performs the best. However, for Q=3 in Figure 10(c), the user-oriented greedy algorithm has better throughput for values of α up to 0.3. For Q=4 in Figure 10(d), the greedy algorithm performs better up to α=0.7. This occurs because although the maxmatch algorithm is guaranteed to produce a schedule with maximum possible connectivity, it may do so at the expense of individual slot performance, since the probability of correct packet reception differs in the slots due to differing levels of other-user interference. When other-user interference is moderate to high, the greedy algorithm may, due to fewer schedulings, perform better than the

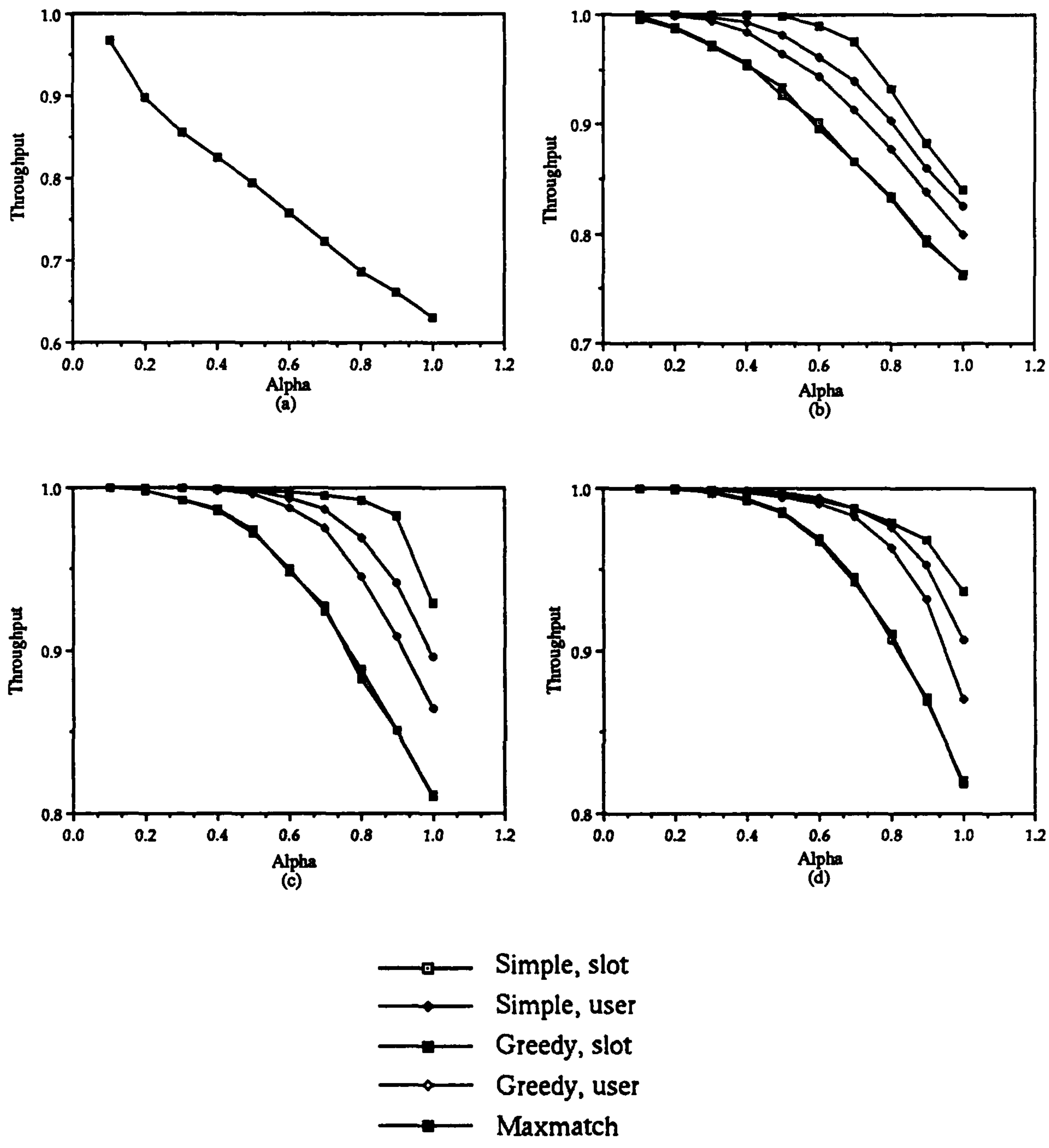


Figure 10. Throughput vs. Alpha for Single Reception Systems, d=1.0.
(a) Q=1. (b) Q=2. (c) Q=3. (d) Q=4.

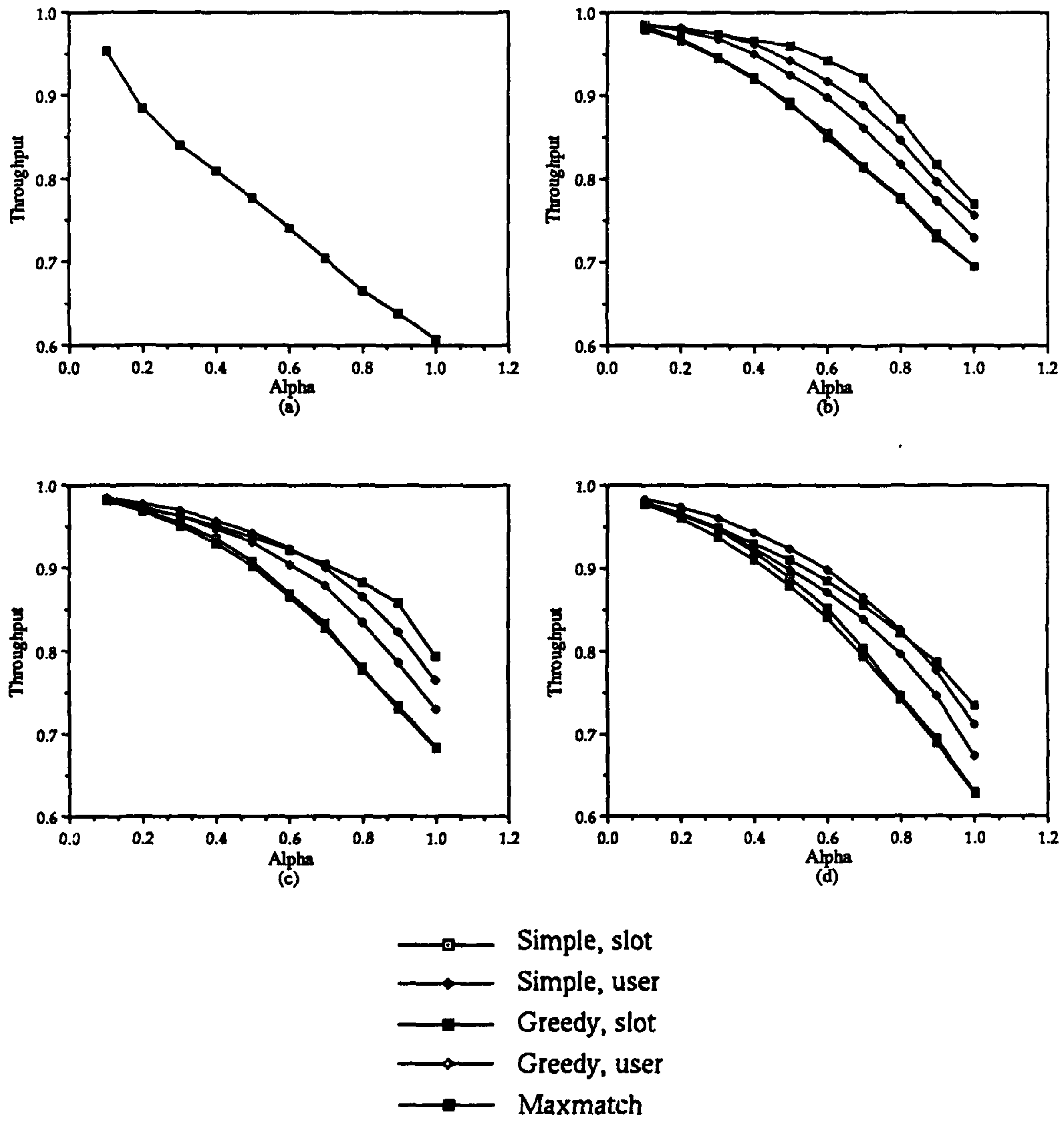


Figure 11. Throughput vs. Alpha for Single Reception Systems, d=0.9.
(a) Q=1. (b) Q=2. (c) Q=3. (d) Q=4.

maxmatch algorithm. Of course, when the system load gets too large, the maxmatch algorithm is best able to cope with the more complex graph, giving it better performance for the highest values of α. When the greedy algorithm does perform better, though, note that the increase in throughput is less than 10^{-3}.

Figure 11 shows the performance of the same systems with degradation introduced by setting d=0.9. Again, the Q=1 case gives equivalent performance, as seen in Figure 11(a), and the order of increasing throughput among the five algorithms for higher values of Q is the same as that for d=1.0. The greedy algorithm also outperforms the maxmatch algorithm for moderate values of α. This occurs, however, up to α=0.6 for the Q=3 case in Figure 11(c) and up to α=0.8 for the Q=4 case in Figure 11(d). The increase in throughput can also be as high as 0.007 for the Q=3 case and 0.014 for the Q=4 case, considerably higher than the increases observed for the d=1.0 case. This is due to the greedy algorithm's better showing in the presence of moderate to high interference, and generally, when there is more interference, there is a greater gain over an algorithm such as the maxmatch algorithm that is relatively blind to such interference. The maxmatch algorithm eventually performs better for the highest values of α, as before, since the graph representing the system becomes too complex for the greedy algorithm to maintain its advantage.

B. Performance of Combining Reception Systems

The comparison made for the combining reception algorithms is between the simple and greedy algorithms, each only using the slot-oriented method for convenience as explained in Chapter III, or the balancing algorithm. With the capability of combining receptions, the additional method of incremental redundancy is also explored in comparison to traditional retransmission.

1. Traditional retransmission

The plots for this case are found in Figures 12 and 13. Again, there are four graphs in each figure corresponding to the differing values of Q used. Figure 12 represents the case of no degradation, or d=1.0. The performances in the Q=1 case are equivalent as before, and the greedy algorithm easily outperforms the simple algorithm in all other cases, as before. In fact, note the steady, almost linear, decline in performance for the simple algorithm in each case, and the degradation in performance as Q increases. Clearly, the random choices of the simple algorithm are not suited well to the multiple receptions each user can have. No exploitation of past user performance is used for the choices made. Therefore, as there are more edges in the graph representation of the system, the performance decreases until the 0.5 throughput boundary is actually crossed in the Q=4 case illustrated in Figure 12(d).

The balancing algorithm, the most meticulous algorithm of the three, performs quite well. However, just as it did with the maxmatch algorithm, the greedy algorithm proves that it can improve the balancing algorithm's performance under certain conditions. In particular, note the cases of Q=3 in Figure 12(c) and Q=4 in Figure 12(d). The balancing algorithm exhibits a near-constant performance for a large range of α, staying above 0.98 until α=0.9, showing great versatility in the presence of complex graphs. The greedy algorithm, meanwhile, exhibits a more graceful decline in performance with increasing α, yet it still holds a high value of throughput for a moderate range of α.

This difference in the two algorithm's characteristics provides the basis for their competition for better throughput. The greedy algorithm outperforms the balancing

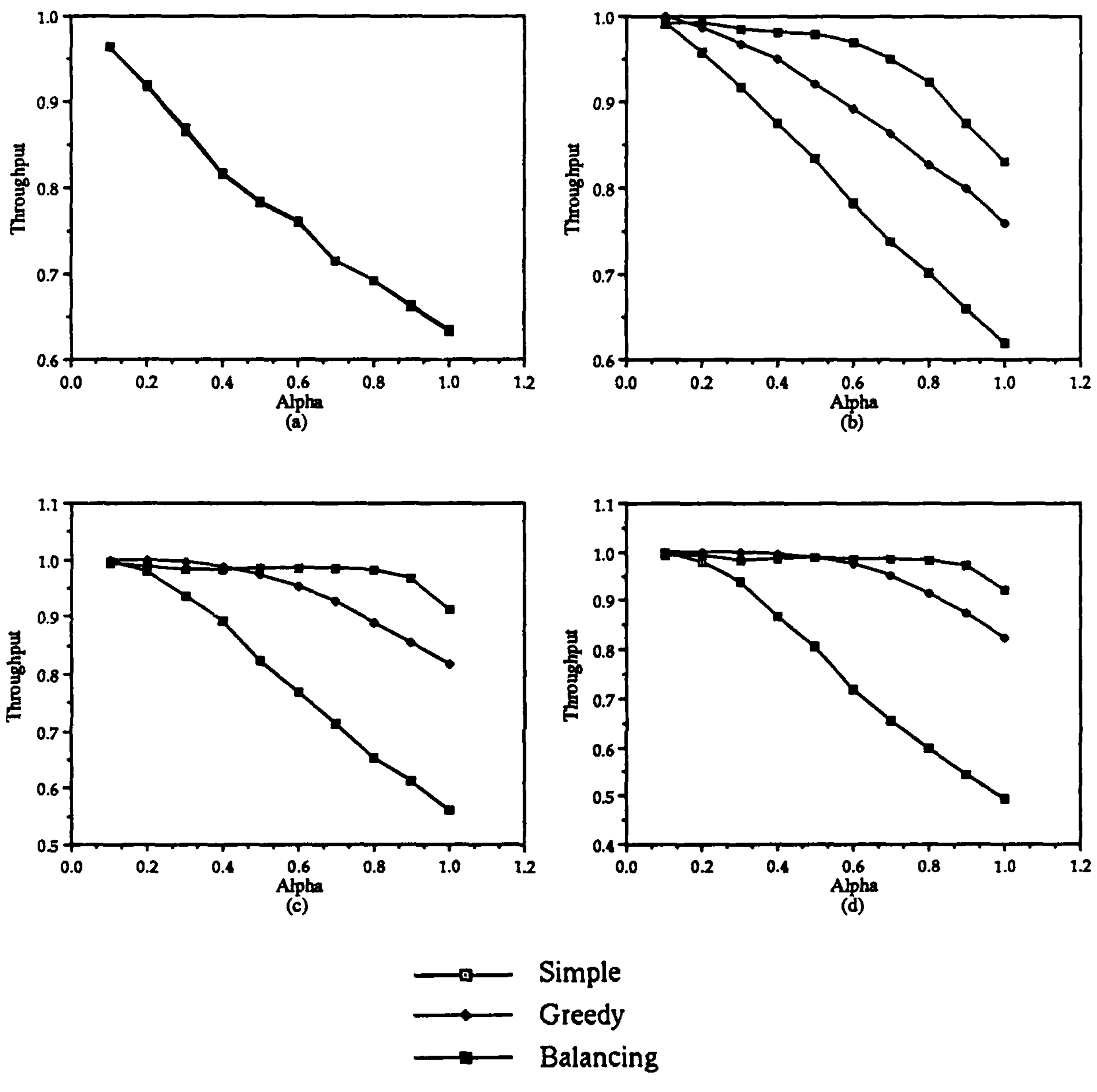


Figure 12. Throughput vs. Alpha for Combining Reception Systems Using Traditional Retransmission, d=1.0. (a) Q=1. (b) Q=2. (c) Q=3. (d) Q=4.

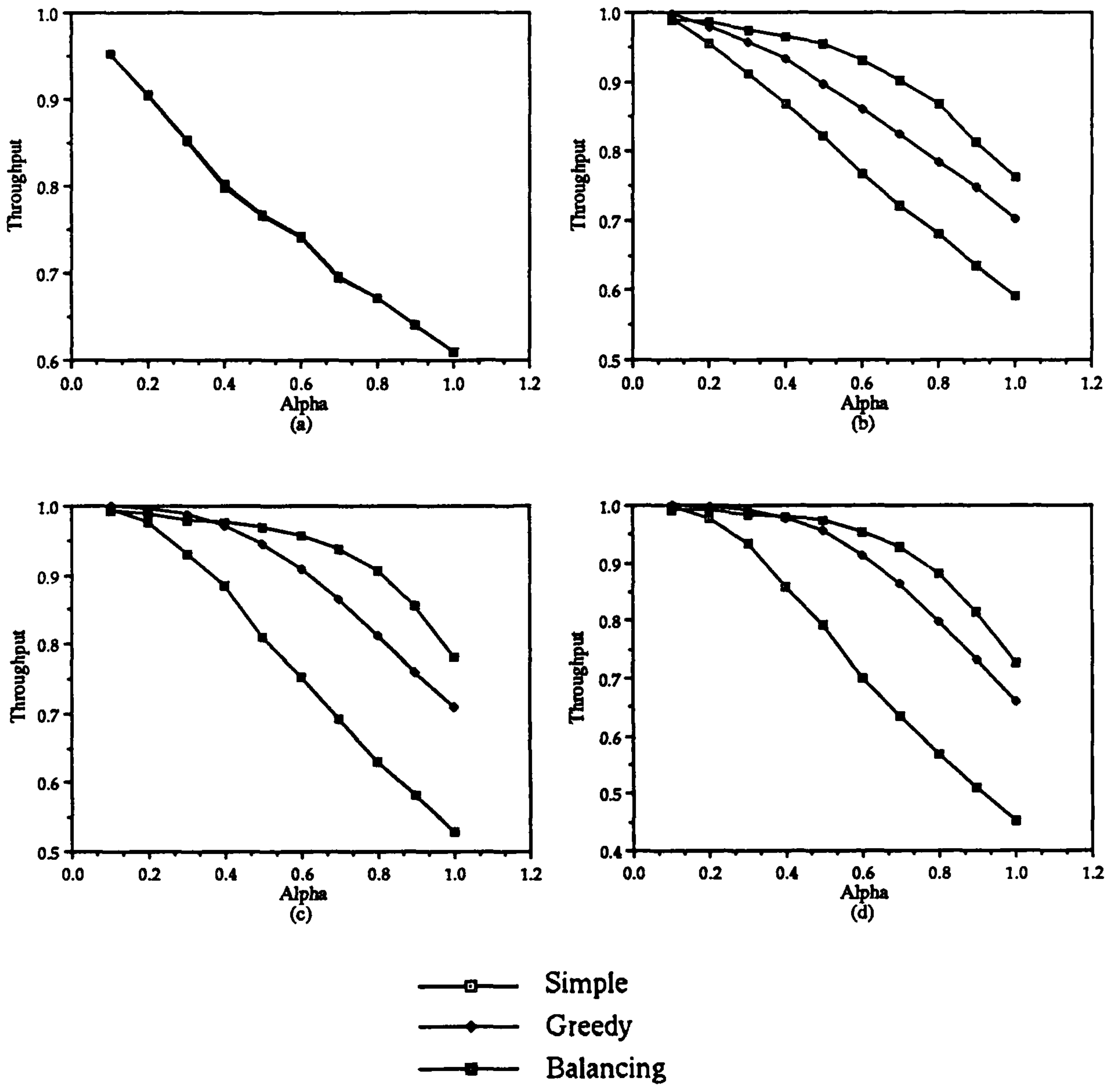


Figure 13. Throughput vs. Alpha for Combining Reception Systems Using Traditional Retransmission, d=0.9. (a) Q=1. (b) Q=2. (c) Q=3. (d) Q=4.

algorithm for lower values of α, up to α=0.4 for Q=3 and up to α=0.5 for Q=4. The greedy algorithm's advantage is not as dramatic as its advantage over the maxmatch algorithm in the single reception case, but it is still a notable result. It occurs not because the balancing algorithm is "blind" to the other-user interference level of slots as the maxmatch algorithm was, since the balancing algorithm uses these levels as the basis for computing a schedule. For low values of α, the balancing algorithm unnecessarily balances out levels for the scheduled users when it would perform well with just scheduling as many users as possible, which is what the greedy algorithm attempts to do. In other words, for these lower values of α, it is more important to combat primary interference than the secondary interference since the levels of other-user interference are not that high. When more users are in the system, balancing helps to counteract primary as well as secondary interference in a more pronounced manner than when α is not as large, allowing the balancing algorithm to outperform the greedy algorithm.

This curious effect is also seen in the near-perfect performance of the greedy algorithm when α=0.1, a result the balancing algorithm cannot match since it is balancing while scheduling rather than just scheduling needed users. In this case, even the simple algorithm can have better performance than the balancing algorithm. The advantage of the greedy algorithm is often small, but it can occasionally increase the throughput by as much as 0.015, which occurs when α=0.3 for the Q=3 and Q=4 cases.

Similar results hold in the case of more degradation, which is illustrated in Figure 13 for d=0.9. There are only a few differences between this case and the previous one. The declines in performance of the greedy algorithm are less graceful and more linear, and the balancing algorithm performance is less constant, indicating less versatility in the presence of increased background noise. The balancing algorithm throughput does stay above 0.95 until α=0.7, though, and it is the choice when there are more users, since the aforementioned greedy algorithm advantage occurs here only until α=0.3 for both the Q=3 and Q=4 cases. The advantage is less pronounced since there is greater overall background noise for all levels of α, making balancing more useful at an earlier value of α.

2. Incremental redundancy

The results of the incremental redundancy case show many of the characteristics of the traditional retransmission case, but there is one important difference: in all cases using incremental redundancy, a decline in error due to secondary interference and a corresponding gain in overall throughput were observed. The three algorithms are again equivalent for the Q=1 case, and the simple algorithm exhibits poor performance, getting worse with increasing Q and decreasing almost linearly in throughput with increasing α.

Figure 14 illustrates the cases with no degradation. Again, the greedy algorithm exhibits a graceful decline with increasing α, while the balancing algorithm retains a high throughput value for a large range of α, staying above 0.98 until α=0.9. Also, the greedy algorithm shows better throughput than the balancing algorithm under certain conditions: up to α=0.4 for Q=3, as in Figure 14(c), and up to α=0.5 for Q=4, as in Figure 14(d). The advantage is small, but again it is as high as 0.015 for α=0.3. The greedy algorithm exhibits perfect performance again for α=0.1 when Q is two or larger. These results are identical to those found using traditional retransmission, but these cases do prove that incremental redundancy can increase throughput, even if it is by no more than 0.01 in every case except Q=4 and α=1.0.

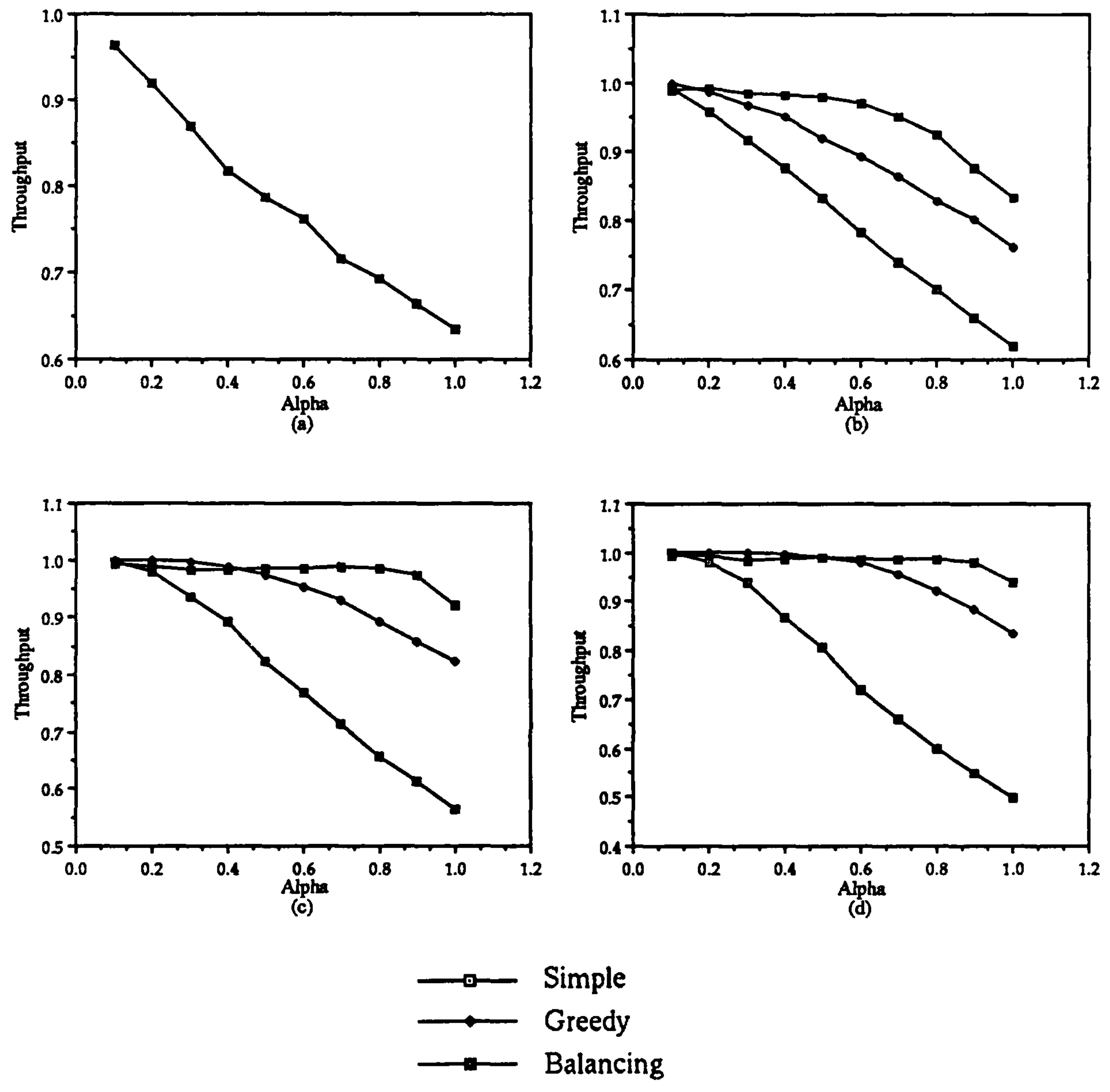


Figure 14. Throughput vs. Alpha for Combining Reception Systems Using Incremental Redundancy, d=1.0. (a) Q=1. (b) Q=2. (c) Q=3. (d) Q=4.

Figure 15 shows the results of the system when d=0.9. Incremental redundancy has a more profound effect here since there is a greater level of background noise. The plots of throughput for the algorithms fall off faster than they did when there was no degradation, but notice how stable the plot for the balancing algorithm is for the Q=4 case in Figure 15(d). In this case, throughput stays above 0.97 until α=0.7, which is the best performance of any algorithm in the presence of such a degraded system. The greedy algorithm again betters the balancing algorithm for a low range of α for the Q=3 and Q=4 cases: up to α=0.3 for Q=3, as in Figure 15(c), up to α=0.4 for Q=4, as in Figure 15(d), and always at the α=0.1 case when Q is greater than one. Here the greedy algorithm throughput advantage is small, touching 0.014 only a couple of times. The advantage of using incremental redundancy is more evident for d=0.9 than it was with no degradation, as it often gives an advantage in throughput of 0.02 over the traditional retransmission case. The advantage occasionally goes as high as 0.05, showing how incremental redundancy helps more in cases with background noise.

C. Evaluation of Relative Effects of Primary and Secondary Interferences

There are further insights to be gained from looking at the individual error rates due to both primary and secondary interferences, as explained in Chapter IV. These data are presented in Tables 2 through 5. Each table contains six columns of data, two columns each corresponding to the simple, greedy and balancing algorithms. As with the graphical data, each table varies with Q and α. The primary error columns show the fraction of users that were never scheduled, and the total error columns show the fraction of symbols that were received incorrectly due to both primary and secondary interferences.

Table 2 illustrates the case using traditional retransmission and no degradation. For almost every case, note how the total error rate is only a little higher than the primary error rate, indicating that any increases in performance must first be achieved by lowering the primary interference. One way this can occur, if one is not using the simple algorithm, is to increase Q, since for both the greedy and balancing algorithms, the best performance occurs for Q=4. If one is using the simple algorithm, better performance would immediately come from using another algorithm. Another way to increase performance would be to lower the value of α, presumably by increasing the number of slots per frame or by decreasing the offered traffic to the system by decreasing the number of users. The only condition where the secondary interference is significant, relative to the primary interference, is when the balancing algorithm is used with the most complex of graphs, as when Q=4 and α=1.0. Even in this case, though, the secondary interference is lower than the primary, so that one is always guaranteed of receiving packets of high quality if the packets are successfully transmitted. The additional Reed-Solomon coding also helps to insure this, as transmissions utilizing this type of coding will either be received with high quality or be completely incorrect.

Table 3 indicates the same systems when additional degradation is introduced, with d=0.9. The Q=1 and Q=2 cases are much like the previous case in that the secondary interference is still small, although the balancing algorithm shows high susceptibility to the higher background noise relative to the low primary interference achieved. For all cases of the simple algorithm, the primary interference always drowns out the possible gains of coding. For the Q=3 and Q=4 cases of the other two algorithms, secondary interference begins to play a larger role due to the added background noise. In fact, increasing Q to combat primary interference may not even be advisable, since the Q=3 case clearly produces the best performance. This occurs since the system benefits from having fewer transmissions per user, and correspondingly, less total interference per slot, when the additional background noise is present. Decreasing α may still be advisable, though. Note

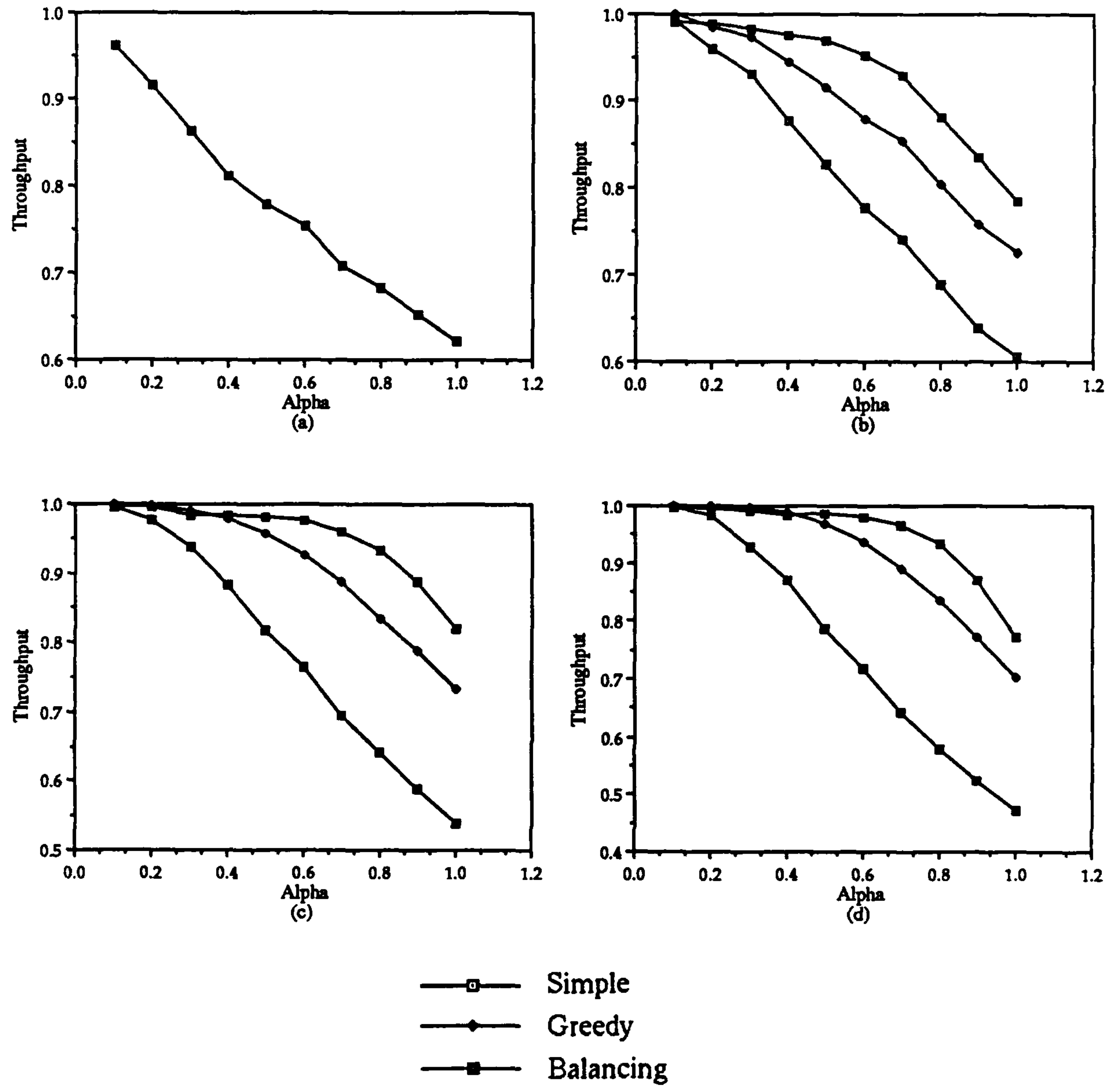


Figure 15. Throughput vs. Alpha for Combining Reception Systems using Incremental Redundancy, d=0.9. (a) Q=1. (b) Q=2. (c) Q=3. (d) Q=4.

Table 2. Comparison of Primary and Secondary Interferences with Traditional Retransmission, d=1.0.

		SIMPLE		GREEDY		BALANCING	
	ALPHA	PRIMARY	TOTAL	PRIMARY	TOTAL	PRIMARY	TOTAL
Q=1	0.1	0.036000	0.036000	0.036000	0.036000	0.036000	0.036000
	0.2	0.081000	0.081002	0.081000	0.081002	0.082500	0.082502
	0.3	0.131000	0.131009	0.131000	0.131009	0.133667	0.133676
	0.4	0.181750	0.181784	0.181750	0.181784	0.185000	0.185033
	0.5	0.214400	0.214454	0.214400	0.214454	0.216800	0.216854
	0.6	0.239333	0.239391	0.239333	0.239391	0.241167	0.241224
	0.7	0.284286	0.284391	0.284286	0.284391	0.285286	0.285391
	0.8	0.307375	0.307522	0.307375	0.307522	0.308500	0.308643
	0.9	0.337111	0.337335	0.337111	0.337335	0.338000	0.338221
	1.0	0.365900	0.366183	0.365900	0.366183	0.366700	0.366983
Q=2	0.1	0.007000	0.007000	0.000000	0.000001	0.010000	0.010000
	0.2	0.041500	0.041501	0.013500	0.013511	0.008500	0.008502
	0.3	0.082000	0.082010	0.032333	0.032419	0.015333	0.015364
	0.4	0.124200	0.124275	0.049250	0.049397	0.018750	0.018891
	0.5	0.166000	0.166117	0.079400	0.079719	0.020400	0.020639
	0.6	0.217500	0.217637	0.106667	0.107352	0.030667	0.031461
	0.7	0.260400	0.260675	0.135143	0.136041	0.049714	0.050785
	0.8	0.298500	0.298895	0.171000	0.172443	0.074875	0.076808
	0.9	0.339778	0.340426	0.198000	0.200429	0.121667	0.124615
	1.0	0.380900	0.381703	0.237300	0.240145	0.165300	0.168621
Q=3	0.1	0.003000	0.003000	0.000000	0.000000	0.006000	0.006000
	0.2	0.022000	0.022002	0.001500	0.001503	0.012000	0.012000
	0.3	0.063000	0.063011	0.004000	0.004049	0.019667	0.019667
	0.4	0.108250	0.108300	0.012750	0.012955	0.017500	0.017504
	0.5	0.176800	0.176930	0.025400	0.026014	0.015000	0.015032
	0.6	0.231000	0.231444	0.045667	0.047415	0.016000	0.016170
	0.7	0.287143	0.287699	0.068571	0.072109	0.013429	0.014199
	0.8	0.344875	0.345838	0.104125	0.109679	0.014125	0.017540
	0.9	0.385778	0.387630	0.137000	0.146320	0.021667	0.031834
	1.0	0.436500	0.438831	0.169500	0.182329	0.070800	0.086705
Q=4	0.1	0.002000	0.002000	0.000000	0.000000	0.007000	0.007000
	0.2	0.021500	0.021502	0.000500	0.000501	0.009000	0.009000
	0.3	0.062333	0.062351	0.001333	0.001440	0.016333	0.016344
	0.4	0.131500	0.131567	0.003750	0.004259	0.015750	0.015758
	0.5	0.192400	0.192683	0.008800	0.010264	0.011800	0.011833
	0.6	0.279667	0.280134	0.020667	0.024794	0.013667	0.013843
	0.7	0.342000	0.343087	0.038571	0.046894	0.013143	0.014028
	0.8	0.401125	0.402913	0.069250	0.084581	0.013250	0.017005
	0.9	0.451333	0.454402	0.099444	0.123947	0.013778	0.029091
	1.0	0.499800	0.504349	0.140900	0.176086	0.034300	0.076924

Table 3. Comparison of Primary and Secondary Interferences with Traditional Retransmission, d=0.9.

		SIMPLE		GREEDY		BALANCING	
	ALPHA	PRIMARY	TOTAL	PRIMARY	TOTAL	PRIMARY	TOTAL
Q=1	0.1	0.036000	0.048458	0.036000	0.048458	0.036000	0.048458
	0.2	0.081000	0.094543	0.081000	0.094543	0.082500	0.095992
	0.3	0.131000	0.145930	0.131000	0.145930	0.133667	0.148490
	0.4	0.181750	0.198206	0.181750	0.198206	0.185000	0.201298
	0.5	0.214400	0.232016	0.214400	0.232016	0.216800	0.234327
	0.6	0.239333	0.257723	0.239333	0.257723	0.241167	0.259470
	0.7	0.284286	0.304386	0.284286	0.304386	0.285286	0.305340
	0.8	0.307375	0.328442	0.307375	0.328442	0.308500	0.329459
	0.9	0.337111	0.359659	0.337111	0.359659	0.338000	0.360459
	1.0	0.365900	0.389817	0.365900	0.389817	0.366700	0.390562
Q=2	0.1	0.007000	0.009088	0.000000	0.002547	0.010000	0.012257
	0.2	0.041500	0.045950	0.013500	0.020485	0.008500	0.014158
	0.3	0.082000	0.088665	0.032333	0.044372	0.015333	0.025608
	0.4	0.124200	0.133348	0.049250	0.067250	0.018750	0.035615
	0.5	0.166000	0.178393	0.079400	0.103480	0.020400	0.045918
	0.6	0.217500	0.232783	0.106667	0.139764	0.030667	0.069498
	0.7	0.260400	0.278486	0.135143	0.174912	0.049714	0.098226
	0.8	0.298500	0.319556	0.171000	0.216478	0.074875	0.133320
	0.9	0.339778	0.364188	0.198000	0.253355	0.121667	0.187856
	1.0	0.380900	0.408420	0.237300	0.299010	0.165300	0.237944
Q=3	0.1	0.003000	0.003683	0.000000	0.000468	0.006000	0.006355
	0.2	0.022000	0.024824	0.001500	0.004147	0.012000	0.012948
	0.3	0.063000	0.068607	0.004000	0.012653	0.019667	0.022701
	0.4	0.108250	0.116495	0.012750	0.028664	0.017500	0.024083
	0.5	0.176800	0.189139	0.025400	0.055929	0.015000	0.030769
	0.6	0.231000	0.248639	0.045667	0.092291	0.016000	0.044173
	0.7	0.287143	0.308370	0.068571	0.133958	0.013429	0.062528
	0.8	0.344875	0.370292	0.104125	0.186983	0.014125	0.094665
	0.9	0.385778	0.418259	0.137000	0.239190	0.021667	0.144116
	1.0	0.436500	0.472086	0.169500	0.291427	0.070800	0.218303
Q=4	0.1	0.002000	0.002332	0.000000	0.000138	0.007000	0.007225
	0.2	0.021500	0.023573	0.000500	0.001618	0.009000	0.009422
	0.3	0.062333	0.067114	0.001333	0.007504	0.016333	0.017595
	0.4	0.131500	0.140706	0.003750	0.021851	0.015750	0.020938
	0.5	0.192400	0.206398	0.008800	0.043579	0.011800	0.025627
	0.6	0.279667	0.298609	0.020667	0.085924	0.013667	0.044942
	0.7	0.342000	0.367662	0.038571	0.135178	0.013143	0.072653
	0.8	0.401125	0.432535	0.069250	0.201170	0.013250	0.117188
	0.9	0.451333	0.490660	0.099444	0.268997	0.013778	0.185139
	1.0	0.499800	0.546915	0.140900	0.339551	0.034300	0.273537

Table 4. Comparison of Primary and Secondary Interferences with Incremental Redundancy, d=1.0.

		SIMPLE		GREEDY		BALANCING	
	ALPHA	PRIMARY	TOTAL	PRIMARY	TOTAL	PRIMARY	TOTAL
Q=1	0.1	0.036000	0.036000	0.036000	0.036000	0.036000	0.036000
	0.2	0.081000	0.081000	0.081000	0.081000	0.082500	0.082500
	0.3	0.131000	0.131000	0.131000	0.131000	0.133667	0.133667
	0.4	0.181750	0.181755	0.181750	0.181755	0.185000	0.185005
	0.5	0.214400	0.214409	0.214400	0.214409	0.216800	0.216809
	0.6	0.239333	0.239340	0.239333	0.239340	0.241167	0.241173
	0.7	0.284286	0.284304	0.284286	0.284304	0.285286	0.285304
	0.8	0.307375	0.307412	0.307375	0.307412	0.308500	0.308537
	0.9	0.337111	0.337174	0.337111	0.337174	0.338000	0.338063
	1.0	0.365900	0.365985	0.365900	0.365985	0.366700	0.366785
Q=2	0.1	0.007000	0.007000	0.000000	0.000000	0.010000	0.010000
	0.2	0.041500	0.041500	0.013500	0.013501	0.008500	0.008500
	0.3	0.082000	0.082001	0.032333	0.032358	0.015333	0.015339
	0.4	0.124200	0.124252	0.049250	0.049285	0.018750	0.018795
	0.5	0.166000	0.166031	0.079400	0.079492	0.020400	0.020458
	0.6	0.217500	0.217530	0.106667	0.106907	0.030667	0.030957
	0.7	0.260400	0.260496	0.135143	0.135437	0.049714	0.050063
	0.8	0.298500	0.298638	0.171000	0.171592	0.074875	0.075664
	0.9	0.339778	0.340051	0.198000	0.199126	0.121667	0.123036
	1.0	0.380900	0.381213	0.237300	0.238576	0.165300	0.166785
Q=3	0.1	0.003000	0.003000	0.000000	0.000000	0.006000	0.006000
	0.2	0.022000	0.022000	0.001500	0.001500	0.012000	0.012000
	0.3	0.063000	0.063001	0.004000	0.004007	0.019667	0.019667
	0.4	0.108250	0.108259	0.012750	0.012803	0.017500	0.017500
	0.5	0.176800	0.176831	0.025400	0.025585	0.015000	0.015004
	0.6	0.231000	0.231191	0.045667	0.046400	0.016000	0.016035
	0.7	0.287143	0.287344	0.068571	0.070260	0.013429	0.013635
	0.8	0.344875	0.345303	0.104125	0.106926	0.014125	0.015703
	0.9	0.385778	0.386731	0.137000	0.142281	0.021667	0.027297
	1.0	0.436500	0.437746	0.169500	0.176886	0.070800	0.080110
Q=4	0.1	0.002000	0.002000	0.000000	0.000000	0.007000	0.007000
	0.2	0.021500	0.021500	0.000500	0.000500	0.009000	0.009000
	0.3	0.062333	0.062338	0.001333	0.001363	0.016333	0.016337
	0.4	0.131500	0.131514	0.003750	0.003936	0.015750	0.015750
	0.5	0.192400	0.192511	0.008800	0.009430	0.011800	0.011813
	0.6	0.279667	0.279827	0.020667	0.022769	0.013667	0.013689
	0.7	0.342000	0.342508	0.038571	0.043151	0.013143	0.013288
	0.8	0.401125	0.402023	0.069250	0.078490	0.013250	0.014294
	0.9	0.451333	0.453033	0.099444	0.114961	0.013778	0.021150
	1.0	0.499800	0.502416	0.140900	0.164625	0.034300	0.063088

Table 5. Comparison of Primary and Secondary Interferences with Incremental Redundancy, d=0.9.

		SIMPLE		GREEDY		BALANCING	
	ALPHA	PRIMARY	TOTAL	PRIMARY	TOTAL	PRIMARY	TOTAL
Q=1	0.1	0.036000	0.038875	0.036000	0.038875	0.036000	0.038851
	0.2	0.081000	0.084660	0.081000	0.084660	0.082500	0.086128
	0.3	0.131000	0.135709	0.131000	0.135709	0.133667	0.138361
	0.4	0.181750	0.187651	0.181750	0.187651	0.185000	0.190872
	0.5	0.214400	0.221237	0.214400	0.221237	0.216800	0.223622
	0.6	0.239333	0.246777	0.239333	0.246777	0.241167	0.248602
	0.7	0.284286	0.293143	0.284286	0.293143	0.285286	0.294135
	0.8	0.307375	0.317046	0.307375	0.317046	0.308500	0.318166
	0.9	0.337111	0.348078	0.337111	0.348078	0.338000	0.348949
	1.0	0.365900	0.378045	0.365900	0.378045	0.366700	0.378840
Q=2	0.1	0.007000	0.007422	0.000000	0.000626	0.010000	0.010497
	0.2	0.041500	0.042830	0.013500	0.016116	0.008500	0.010280
	0.3	0.082000	0.084483	0.032333	0.037355	0.015333	0.018744
	0.4	0.124200	0.128115	0.049250	0.057408	0.018750	0.025751
	0.5	0.166000	0.171028	0.079400	0.092330	0.020400	0.034077
	0.6	0.217500	0.224024	0.106667	0.124623	0.030667	0.051537
	0.7	0.260400	0.268422	0.135143	0.157631	0.049714	0.076823
	0.8	0.298500	0.309580	0.171000	0.200249	0.074875	0.111642
	0.9	0.339778	0.352614	0.198000	0.233160	0.121667	0.164916
	1.0	0.380900	0.396473	0.237300	0.278430	0.165300	0.214387
Q=3	0.1	0.003000	0.003151	0.000000	0.000237	0.006000	0.006072
	0.2	0.022000	0.022740	0.001500	0.002622	0.012000	0.012164
	0.3	0.063000	0.064905	0.004000	0.008155	0.019667	0.020330
	0.4	0.108250	0.111910	0.012750	0.022679	0.017500	0.019667
	0.5	0.176800	0.182661	0.025400	0.043662	0.015000	0.020433
	0.6	0.231000	0.239056	0.045667	0.074875	0.016000	0.027028
	0.7	0.287143	0.298667	0.068571	0.112255	0.013429	0.039297
	0.8	0.344875	0.359453	0.104125	0.161930	0.014125	0.066232
	0.9	0.385778	0.403842	0.137000	0.212439	0.021667	0.110339
	1.0	0.436500	0.456367	0.169500	0.259800	0.070800	0.182201
Q=4	0.1	0.002000	0.002187	0.000000	0.000094	0.007000	0.007157
	0.2	0.021500	0.022048	0.000500	0.000847	0.009000	0.009166
	0.3	0.062333	0.064349	0.001333	0.004201	0.016333	0.016535
	0.4	0.131500	0.135246	0.003750	0.012154	0.015750	0.016319
	0.5	0.192400	0.199615	0.008800	0.030699	0.011800	0.014730
	0.6	0.279667	0.290670	0.020667	0.060542	0.013667	0.022733
	0.7	0.342000	0.357235	0.038571	0.106464	0.013143	0.037052
	0.8	0.401125	0.418363	0.069250	0.166204	0.013250	0.069107
	0.9	0.451333	0.474355	0.099444	0.225817	0.013778	0.130599
	1.0	0.499800	0.528936	0.140900	0.299303	0.034300	0.226820

that switching to the balancing algorithm is less effective due to the additional background noise. In this case, the secondary error rate often overwhelms the primary error rate, which makes this algorithm useful only in a system requiring high primary throughput and moderate secondary throughput, such as packetized voice. Any additional increases in throughput must occur by lowering the secondary interference. One way this can be achieved is through the use of incremental redundancy.

Table 4 shows the analysis of the systems while utilizing incremental redundancy with no degradation. Using this coding allows roughly similar performance to the traditional retransmission case, although the gains achieved allow use of the next echelon of α. For instance, the amount of secondary error at α=0.5 is comparable to that of the system with traditional retransmission at α=0.4. As before, the solutions to lowering the primary interference are moving to a better algorithm, increasing Q and lowering α. The best performance again occurs for Q=4, and lowering α may not be as necessary as it was with traditional retransmission, as discussed above. Still, the increases in performance over traditional retransmission are limited to no more than 0.01 in almost all cases, but it is clear that without significant system changes, improvements can be made by using incremental redundancy with the system.

Table 5 shows the previous incremental redundancy system with d=0.9. The additional coding allows one to use a value of α that is several echelons higher, as opposed to the gain of one echelon when d=1.0. The balancing algorithm also shows less susceptibility to the additional background noise than without incremental redundancy. Again, the best performance occurs at Q=3 due to the additional noise. As in the initial analysis of this system, incremental redundancy can improve performance by as much as 0.05 over the system that does not use it.

In all the cases where the greedy algorithm outperforms the balancing algorithm, the greedy algorithm is more sensitive to secondary interference, which is expected when the primary interference is smaller. It is only at the crossover points of α where the balancing algorithm begins to do outperform the greedy algorithm, however, that the additional secondary interference incurred by the greedy system causes it to perform worse. Otherwise, it is the poorer primary interference of the greedy algorithm that increases the total error. It is reasonable to conclude, though, that the balancing algorithm is well mated to the incremental redundancy method, since it incurs less of a penalty on the throughput by the secondary interference than does the greedy algorithm.

VI. CONCLUSIONS

In this thesis several new methods have been found to improve the performance of solutions to the scheduling problem for the silent receiver. These include combining receptions with incremental redundancy, using balancing algorithms, and implementing algorithms in real time. Brief guidelines for employing these methods are given in this chapter.

In the single reception case, the maxmatch algorithm handles scheduling very well, if the complexity is not a problem. It also is the choice if high performance is needed in a complex system. However, the user-oriented greedy algorithm performs comparably, if not better, under most other conditions, such as in the presence of background noise and low-to-moderate load. In addition, the greedy algorithm operates with a lower complexity, making it much faster.

In the combining reception case with no incremental redundancy, the balancing algorithm exhibits high performance over a wide range of loads. It also is the best at dealing with secondary interference, especially when the system load is moderate to high. The greedy algorithm will perform best when the load is low; with a lower complexity than the balancing algorithm, it may be the algorithm of choice under these conditions. In the presence of background noise, the balancing algorithm is even more preferable due to its superior ability to handle secondary interference. It also retains many of the characteristics displayed when there is no background noise. The greedy algorithm is preferable only if the load is quite low.

Performance can be improved by utilizing incremental redundancy. The improvement in performance may justify the use of a less optimal scheduling algorithm such as the less complex greedy algorithm. Incremental redundancy allows one to load the system considerably more and retain the same performance, even when there is background noise. The balancing algorithm is best suited to utilize the advantages offered by incremental redundancy, since both can be used specifically to combat secondary interference.

Another option for increasing performance includes finding the optimal value for the rebroadcast redundancy. Simply increasing the rebroadcast redundancy may not be the answer, as demonstrated in the case where there was background noise. Thus, the optimal value for the rebroadcast redundancy involves balancing the effects of secondary interference and primary interference. One may also consider lowering the load on the system by lowering α. It is clear, however, that most gains are first found by lowering the levels of primary interference, since successfully scheduled packets are often of high quality in a variety of circumstances. Lower primary interference is then the most convincing argument for the balancing algorithm.

It is easy to see why the greedy algorithm would be the algorithm of choice in the real time case, but one should also consider the advantages of using incremental redundancy. In real time, transmission failure information is readily available. This information allows for the adaptation of future scheduling decisions to account for lost symbols. These modifications will increase performance and certainly warrant the use of incremental redundancy, which can utilize fractions of packets rather than request complete additional broadcasts. Incremental redundancy also adds only to the coding, so that the complexity of the scheduling algorithm is not considerably increased. The coding and decoding may be difficult in real time, but the suspected additional gains in performance certainly justify its use.

REFERENCES

[1] R. Cruz and B. Hajek, "Global load balancing by local adjustments," *Proceedings of the 19th Annual Conference on Information Sciences and Systems*, Johns Hopkins University, pp. 448-454, 1985.

[2] B. Hajek, "Balanced scheduling in a packet synchronized spread spectrum network," *Proceedings 1983 IEEE INFOCOM*, pp. 56-65, 1983.

[3] M.B. Pursley and S.D.Sandberg, "Simulation of delay in frequency-hop packet radio networks," *Proceedings of the 21st Annual Conference on Information Sciences and Systems*, Johns Hopkins University, March 1987.

[4] B. Hajek, "Asymptotic analysis of an assignment problem arising in a distributed communications protocol," *Proceedings of the 1988 Conference on Decision and Control*, 1988.

[5] C.P. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Englewood Cliffs, NJ: Prentice Hall, 1982.

[6] J.E. Hopcroft and R.M. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs," *SIAM Journal of Computing*, vol. 2, no. 4, pp. 225-231, December 1973.

[7] M.B. Pursley, "Packet error probabilities in frequency-hop radio networks--Coping with statistical dependence and noisy side information," *IEEE Global Telecommunications Conference Record*, vol. 1, pp. 165-170, December 1986.

[8] J.E. Wieselthier and A. Ephremides, "A distributed reservation scheme for spread spectrum multiple access channels," *Proceedings of 1983 GLOBECOM*, pp. 659-665.

[9] J.E. Wieselthier, A. Ephremides and J.A.B. Tarr, "A distributed reservation-based CDMA protocol that does not require feedback information," *IEEE Transactions on Communications*, vol. COM-36, pp. 913-923, August 1988.